

[Overview](#) [Package](#) [Class Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

Java™ 2 Platform

Std. Ed. v1.4.2

[PREV CLASS](#) [NEXT CLASS](#)[FRAMES](#) [NO FRAMES](#) [All Classes](#)SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

java.util

Interface Map

All Known Subinterfaces:

[SortedMap](#)

All Known Implementing Classes:

[AbstractMap](#), [Attributes](#), [HashMap](#), [Hashtable](#), [IdentityHashMap](#), [RenderingHints](#), [TreeMap](#),
[WeakHashMap](#)

public interface **Map**

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

This interface takes the place of the `Dictionary` class, which was a totally abstract class rather than an interface.

The `Map` interface provides three *collection views*, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings. The *order* of a map is defined as the order in which the iterators on the map's collection views return their elements. Some map implementations, like the `TreeMap` class, make specific guarantees as to their order; others, like the `HashMap` class, do not.

Note: great care must be exercised if mutable objects are used as map keys. The behavior of a map is not specified if the value of an object is changed in a manner that affects equals comparisons while the object is a key in the map. A special case of this prohibition is that it is not permissible for a map to contain itself as a key. While it is permissible for a map to contain itself as a value, extreme caution is advised: the equals and hashCode methods are no longer well defined on a such a map.

All general-purpose map implementation classes should provide two "standard" constructors: a void (no arguments) constructor which creates an empty map, and a constructor with a single argument of type `Map`, which creates a new map with the same key-value mappings as its argument. In effect, the latter constructor allows the user to copy any map, producing an equivalent map of the desired class. There is no way to enforce this recommendation (as interfaces cannot contain constructors) but all of the general-purpose map implementations in the SDK comply.

The "destructive" methods contained in this interface, that is, the methods that modify the map on which they operate, are specified to throw `UnsupportedOperationException` if this map does not support the operation. If this is the case, these methods may, but are not required to, throw an `UnsupportedOperationException` if the invocation would have no effect on the map. For example, invoking the [putAll\(Map\)](#) method on an unmodifiable map may, but is not required to, throw the exception if the map whose mappings are to be "superimposed" is empty.

Some map implementations have restrictions on the keys and values they may contain. For example, some implementations prohibit null keys and values, and some have restrictions on the types of their keys. Attempting to insert an ineligible key or value throws an unchecked exception, typically

`NullPointerException` or `ClassCastException`. Attempting to query the presence of an ineligible key or value may throw an exception, or it may simply return false; some implementations will exhibit the former behavior and some will exhibit the latter. More generally, attempting an operation on an ineligible key or value whose completion would not result in the insertion of an ineligible element into the map may throw an exception or it may succeed, at the option of the implementation. Such exceptions are marked as "optional" in the specification for this interface.

This interface is a member of the [Java Collections Framework](#).

Since:

1.2

See Also:

[HashMap](#), [TreeMap](#), [Hashtable](#), [SortedMap](#), [Collection](#), [Set](#)

Nested Class Summary

static interface	Map.Entry A map entry (key-value pair).
------------------	--

Method Summary

void	clear () Removes all mappings from this map (optional operation).
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	entrySet () Returns a set view of the mappings contained in this map.
boolean	equals (Object o) Compares the specified object with this map for equality.
Object	get (Object key) Returns the value to which this map maps the specified key.
int	hashCode () Returns the hash code value for this map.
boolean	isEmpty () Returns true if this map contains no key-value mappings.
Set	keySet () Returns a set view of the keys contained in this map.
Object	put (Object key, Object value) Associates the specified value with the specified key in this map (optional operation).
void	putAll (Map t) Copies all of the mappings from the specified map to this map (optional operation).
Object	remove (Object key) Removes the mapping for this key from this map if it is present (optional operation).
int	size () Returns the number of key-value mappings in this map.

Collection	values () Returns a collection view of the values contained in this map.
----------------------------	---

Method Detail

size

```
public int size()
```

Returns the number of key-value mappings in this map. If the map contains more than `Integer.MAX_VALUE` elements, returns `Integer.MAX_VALUE`.

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

isEmpty

```
public boolean isEmpty()
```

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

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

containsKey

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

Returns `true` if this map contains a mapping for the specified key. More formally, returns `true` if and only if this map contains a mapping for a key `k` such that `(key==null ? k==null : key.equals(k))`. (There can be at most one such mapping.)

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 key is of an inappropriate type for this map (optional).
[NullPointerException](#) - if the key is `null` and this map does not permit `null` keys (optional).

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 of the `Map` interface.

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.

Throws:

[ClassCastException](#) - if the value is of an inappropriate type for this map (optional).

[NullPointerException](#) - if the value is `null` and this map does not permit `null` values (optional).

get

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

Returns the value to which this map maps the specified key. Returns `null` if the map contains no mapping for this key. 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.

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.)

Parameters:

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

Returns:

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

Throws:

[ClassCastException](#) - if the key is of an inappropriate type for this map (optional).

[NullPointerException](#) - key is `null` and this map does not permit `null` keys (optional).

See Also:

[containsKey\(Object\)](#)

put

```
public Object put(Object key,  
                  Object value)
```

Associates the specified value with the specified key in this map (optional operation). If the map previously contained a mapping for this key, the old value is replaced by the specified value. (A map `m` is said to contain a mapping for a key `k` if and only if [m.containsKey\(k\)](#) would return `true`.)

Parameters:

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

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

Returns:

previous value associated with specified key, or `null` if there was no mapping for key. A `null`

return can also indicate that the map previously associated `null` with the specified key, if the implementation supports `null` values.

Throws:

[UnsupportedOperationException](#) - if the `put` operation is not supported by this map.

[ClassCastException](#) - if the class of the specified key or value prevents it from being stored in this map.

[IllegalArgumentException](#) - if some aspect of this key or value prevents it from being stored in this map.

[NullPointerException](#) - this map does not permit `null` keys or values, and the specified key or value is `null`.

remove

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

Removes the mapping for this key from this map if it is present (optional operation). More formally, if this map contains a mapping from key `k` to value `v` such that `(key==null ? k==null : key.equals(k))`, that mapping is removed. (The map can contain at most one such mapping.)

Returns the value to which the map previously associated the key, or `null` if the map contained no mapping for this key. (A `null` return can also indicate that the map previously associated `null` with the specified key if the implementation supports `null` values.) The map will not contain a mapping for the specified key once the call returns.

Parameters:

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

Returns:

previous value associated with specified key, or `null` if there was no mapping for key.

Throws:

[ClassCastException](#) - if the key is of an inappropriate type for this map (optional).

[NullPointerException](#) - if the key is `null` and this map does not permit `null` keys (optional).

[UnsupportedOperationException](#) - if the `remove` method is not supported by this map.

putAll

```
public void putAll(Map t)
```

Copies all of the mappings from the specified map to this map (optional operation). The effect of this call is equivalent to that of calling [put\(k, v\)](#) on this map once for each mapping from key `k` to value `v` in the specified map. The behavior of this operation is unspecified if the specified map is modified while the operation is in progress.

Parameters:

`t` - Mappings to be stored in this map.

Throws:

[UnsupportedOperationException](#) - if the `putAll` method is not supported by this map.

[ClassCastException](#) - if the class of a key or value in the specified map prevents it from being stored in this map.

[IllegalArgumentException](#) - some aspect of a key or value in the specified map prevents it

from being stored in this map.

[NullPointerException](#) - the specified map is null, or if this map does not permit null keys or values, and the specified map contains null keys or values.

clear

```
public void clear()
```

Removes all mappings from this map (optional operation).

Throws:

[UnsupportedOperationException](#) - clear is not supported by this map.

keySet

```
public Set 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, 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.

Returns:

a set view of the keys contained in this map.

values

```
public Collection 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, 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.

Returns:

a collection view of the values contained in this map.

entrySet

```
public Set entrySet()
```

Returns a set view of the mappings contained in this map. Each element in the returned set is a [Map.Entry](#). 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, 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.

Returns:

a set view of the mappings contained in this map.

equals

```
public boolean equals(Object o)
```

Compares the specified object with this map for equality. Returns `true` if the given object is also a map and the two Maps represent the same mappings. More formally, two maps `t1` and `t2` represent the same mappings if `t1.entrySet().equals(t2.entrySet())`. This ensures that the `equals` method works properly across different implementations of the `Map` interface.

Overrides:

[equals](#) in class [Object](#)

Parameters:

o - object to be compared for equality with this map.

Returns:

`true` if the specified object is equal to this map.

See Also:

[Object.hashCode\(\)](#), [Hashtable](#)

hashCode

```
public int hashCode()
```

Returns the hash code value for this map. The hash code of a map is defined to be the sum of the hashCodes of each entry in the map's `entrySet` view. This ensures that `t1.equals(t2)` implies that `t1.hashCode()==t2.hashCode()` for any two maps `t1` and `t2`, as required by the general contract of `Object.hashCode`.

Overrides:

[hashCode](#) in class [Object](#)

Returns:

the hash code value for this map.

See Also:

[Map.Entry.hashCode\(\)](#), [Object.hashCode\(\)](#), [Object.equals\(Object\)](#), [equals\(Object\)](#)

[Overview](#) [Package](#) [Class](#) [Use Tree](#) [Deprecated](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

[Submit a bug or feature](#)

For further API reference and developer documentation, see [Java 2 SDK SE Developer Documentation](#). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2003 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](#). Also see the [documentation redistribution policy](#).

*Java™ 2 Platform
Std. Ed. v1.4.2*