Module java.base
Package java.util.concurrent

Interface ConcurrentMap<K,V>

Type Parameters:

K - the type of keys maintained by this map

V - the type of mapped values

All Superinterfaces:

Map<K,V>

All Known Subinterfaces:

ConcurrentNavigableMap<K,V>

All Known Implementing Classes:

ConcurrentHashMap, ConcurrentSkipListMap

```
public interface ConcurrentMap<K,V>
extends Map<K,V>
```

A Map providing thread safety and atomicity guarantees.

To maintain the specified guarantees, default implementations of methods including putIfAbsent(K, V) inherited from Map must be overridden by implementations of this interface. Similarly, implementations of the collections returned by methods Map.keySet(), Map.values(), and Map.entrySet() must override methods such as removeIf when necessary to preserve atomicity guarantees.

Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a ConcurrentMap as a key or value *happen-before* actions subsequent to the access or removal of that object from the ConcurrentMap in another thread.

This interface is a member of the Java Collections Framework.

Since:

1.5

Nested Class Summary

Nested classes/interfaces declared in interface java.util.Map

Map.Entry<K,V>

Method Summary

All Methods Instance Methods Abstract Methods Default Methods

Modifier and Type	Method	Description

default 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).
default 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.
default 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.
default 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.
default 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.
default 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	putIfAbsent(K key, V value)	If the specified key is not already associated with a value, associates it with the given value.
boolean	remove(Object key, Object value)	Removes the entry for a key only if currently mapped to a given value.
V	replace(K key, V value)	Replaces the entry for a key only if currently mapped to some value.
boolean	replace(K key, V oldValue, V newValue)	Replaces the entry for a key only if currently mapped to a given value.
default void	replaceAll(BiFunction<? super K,? super V,? extends	Replaces each entry's value with the result of invoking the given

V> function)	function on that entry until all entries have been processed or the function throws an exception.

Methods declared in interface java.util.Map

clear, containsKey, containsValue, entrySet, equals, get, hashCode, isEmpty, keySet, put, putAll, remove, size, values

Method Details

getOrDefault

```
default 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.

Specified by:

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

Implementation Note:

This implementation assumes that the ConcurrentMap cannot contain null values and `get()` returning null unambiguously means the key is absent. Implementations which support null values **must** override this default implementation.

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

Throws:

`ClassCastException` - if the key is of an inappropriate type for this map (optional)

`NullPointerException` - if the specified key is null and this map does not permit null keys (optional)

Since:

1.8

forEach

```
default 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. 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>`

Implementation Requirements:

The default implementation is equivalent to, for this `map`:

```
for (Map.Entry<K,V> entry : map.entrySet()) {
  action.accept(entry.getKey(), entry.getValue());
}
```

Implementation Note:

The default implementation assumes that `IllegalStateException` thrown by `getKey()` or `getValue()` indicates that the entry has been removed and cannot be processed. Operation continues for subsequent entries.

Parameters:

`action` - The action to be performed for each entry

Throws:

`NullPointerException` - if the specified action is null

Since:

1.8

putIfAbsent

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

If the specified key is not already associated with a value, associates it with the given value. This is equivalent to, for this `map`:

```
if (!map.containsKey(key))
  return map.put(key, value);
else
  return map.get(key);
```

except that the action is performed atomically.

Specified by:

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

Implementation Note:

This implementation intentionally re-abstracts the inappropriate default provided in `Map`.

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.)

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

`NullPointerException` - if the specified key or value is null, and this map does not permit null keys or values

`IllegalArgumentException` - if some property of the specified key or value prevents it from being stored in this map

remove

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

Removes the entry for a key only if currently mapped to a given value. This is equivalent to, for this `map`:

```
if (map.containsKey(key)
    && Objects.equals(map.get(key), value)) {
  map.remove(key);
  return true;
} else {
  return false;
}
```

except that the action is performed atomically.

Specified by:

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

Implementation Note:

This implementation intentionally re-abstracts the inappropriate default provided in `Map`.

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

Throws:

`UnsupportedOperationException` - if the `remove` operation is not supported by this map

`ClassCastException` - if the key or value is of an inappropriate type for this map (optional)

NullPointerException - if the specified key or value is null, and this map does not permit null keys or values (optional)

replace

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

Replaces the entry for a key only if currently mapped to a given value. This is equivalent to, for this map:

```
 if (map.containsKey(key)
     && Objects.equals(map.get(key), oldValue)) {
   map.put(key, newValue);
   return true;
 } else {
   return false;
 }
```

except that the action is performed atomically.

Specified by:

replace in interface Map<K,V>

Implementation Note:

This implementation intentionally re-abstracts the inappropriate default provided in Map.

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

Throws:

UnsupportedOperationException - if the put operation is not supported by this map

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

NullPointerException - if a specified key or value is null, and this map does not permit null keys or values

IllegalArgumentException - if some property of a specified key or value prevents it from being stored in this map

replace

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

Replaces the entry for a key only if currently mapped to some value. This is equivalent to, for this `map`:

```
 if (map.containsKey(key))
   return map.put(key, value);
 else
   return null;
```

except that the action is performed atomically.

Specified by:

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

Implementation Note:

This implementation intentionally re-abstracts the inappropriate default provided in `Map`.

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.)

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

`NullPointerException` - if the specified key or value is null, and this map does not permit null keys or values

`IllegalArgumentException` - if some property of the specified key or value prevents it from being stored in this map

replaceAll

```
default 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. Exceptions thrown by the function are relayed to the caller.

Specified by:

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

Implementation Requirements:

The default implementation is equivalent to, for this `map`:

```
for (Map.Entry<K,V> entry : map.entrySet()) {
  K k;
  V v;
  do {
    k = entry.getKey();
    v = entry.getValue();
  } while (!map.replace(k, v, function.apply(k, v)));
}
```

The default implementation may retry these steps when multiple threads attempt updates including potentially calling the function repeatedly for a given key.

This implementation assumes that the ConcurrentMap cannot contain null values and `get()` returning null unambiguously means the key is absent. Implementations which support null values **must** override this default implementation.

Parameters:

`function` - the function to apply to each entry

Throws:

`UnsupportedOperationException` - if the `set` operation is not supported by this map's entry set iterator.

`NullPointerException` - if function or a replacement value is null, and this map does not permit null keys or values (optional)

`ClassCastException` - if a replacement value is of an inappropriate type for this map (optional)

`IllegalArgumentException` - if some property of a replacement value prevents it from being stored in this map (optional)

Since:

1.8

computeIfAbsent

```
default 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`.

If the mapping function returns `null`, no mapping is recorded. If the mapping 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);
```

The mapping function should not modify this map during computation.

Specified by:

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

Implementation Requirements:

The default implementation is equivalent to the following steps for this `map`:

```
V oldValue, newValue;
return ((oldValue = map.get(key)) == null
        && (newValue = mappingFunction.apply(key)) != null
        && (oldValue = map.putIfAbsent(key, newValue)) == null)
  ? newValue
  : oldValue;
```

This implementation assumes that the ConcurrentMap cannot contain null values and `get()` returning null unambiguously means the key is absent. Implementations which support null values **must** override this default implementation.

Parameters:

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

`mappingFunction` - the mapping 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

Throws:

`UnsupportedOperationException` - if the `put` operation is not supported by this map (optional)

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

`NullPointerException` - if the specified key is null and this map does not support null keys, or the mappingFunction is null

`IllegalArgumentException` - if some property of the specified key or value prevents it from being stored in this map (optional)

Since:

1.8

computeIfPresent

```
default 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.

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

The remapping function should not modify this map during computation.

Specified by:

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

Implementation Requirements:

The default implementation is equivalent to performing the following steps for this `map`:

```
for (V oldValue; (oldValue = map.get(key)) != null; ) {
  V newValue = remappingFunction.apply(key, oldValue);
  if ((newValue == null)
      ? map.remove(key, oldValue)
      : map.replace(key, oldValue, newValue))
    return newValue;
}
return null;
```

When multiple threads attempt updates, map operations and the remapping function may be called multiple times.

This implementation assumes that the ConcurrentMap cannot contain null values and `get()` returning null unambiguously means the key is absent. Implementations which support null values **must** override this default implementation.

Parameters:

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

`remappingFunction` - the remapping function to compute a value

Returns:

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

Throws:

`UnsupportedOperationException` - if the `put` operation is not supported by this map (optional)

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

`NullPointerException` - if the specified key is null and this map does not support null keys, or the remappingFunction is null

`IllegalArgumentException` - if some property of the specified key or value prevents it from being stored in this map (optional)

Since:

1.8

compute

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

The remapping function should not modify this map during computation.

Specified by:

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

Implementation Requirements:

The default implementation is equivalent to performing the following steps for this `map`:

```
for (;;) {
  V oldValue = map.get(key);
  V newValue = remappingFunction.apply(key, oldValue);
  if (newValue != null) {
    if ((oldValue != null)
      ? map.replace(key, oldValue, newValue)
      : map.putIfAbsent(key, newValue) == null)
      return newValue;
  } else if (oldValue == null || map.remove(key, oldValue)) {
    return null;
  }
}
```

When multiple threads attempt updates, map operations and the remapping function may be called multiple times.

This implementation assumes that the ConcurrentMap cannot contain null values and `get()` returning null unambiguously means the key is absent. Implementations which support null values **must** override this default implementation.

Parameters:

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

`remappingFunction` - the remapping function to compute a value

Returns:

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

Throws:

`UnsupportedOperationException` - if the `put` operation is not supported by this map (optional)

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

`NullPointerException` - if the specified key is null and this map does not support null keys, or the remappingFunction is null

`IllegalArgumentException` - if some property of the specified key or value prevents it from being stored in this map (optional)

Since:

1.8

merge

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

The remapping function should not modify this map during computation.

Specified by:

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

Implementation Requirements:

The default implementation is equivalent to performing the following steps for this `map`:

```
for (;;) {
  V oldValue = map.get(key);
  if (oldValue != null) {
    V newValue = remappingFunction.apply(oldValue, value);
    if (newValue != null) {
      if (map.replace(key, oldValue, newValue))
        return newValue;
    } else if (map.remove(key, oldValue)) {
      return null;
    }
  } else if (map.putIfAbsent(key, value) == null) {
    return value;
```

```
    }
  }
```

When multiple threads attempt updates, map operations and the remapping function may be called multiple times.

This implementation assumes that the ConcurrentMap cannot contain null values and `get()` returning null unambiguously means the key is absent. Implementations which support null values **must** override this default implementation.

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 remapping 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

Throws:

`UnsupportedOperationException` - if the `put` operation is not supported by this map (optional)

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

`NullPointerException` - if the specified key is null and this map does not support null keys or the value or remappingFunction is null

`IllegalArgumentException` - if some property of the specified key or value prevents it from being stored in this map (optional)

Since:

1.8
