WeakHashMap Class in Java

Last Updated : 24 Jan, 2022

WeakHashMap is an implementation of the Map interface. WeakHashMap is almost the same as HashMap except in the case of WeakHashMap if the object is specified as the key doesn't contain any references- it is eligible for garbage collection even though it is associated with WeakHashMap. i.e Garbage Collector dominates over WeakHashMap.

WeakHashMap is the Hash table-based implementation of the Map interface, with weak keys. An entry in a WeakHashMap will automatically be removed when its key is no longer in ordinary use. More precisely, the presence of a mapping for a given key will not prevent the key from being discarded by the garbage collector, that is, made finalizable, finalized, and then reclaimed. When a key has been discarded its entry is effectively removed from the map, so this class behaves somewhat differently from other Map implementations.

A few important features of a WeakHashMap Class are:

- Both null values and null keys are supported in WeakHashMap.
- It is not synchronized.
- This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator.

Constructors in WeakHashMap

1. WeakHashMap(): This constructor is used to create an empty WeakHashMap with the default initial capacity-(16) and load factor (0.75).

2. WeakHashMap(int initialCapacity): This constructor is used to create an empty WeakHashMap with the given initial capacity and the default load factor (0.75).

3. WeakHashMap(int initialCapacity, float loadFactor): This constructor is used to create an empty WeakHashMap with the given initial capacity and the given load factor.

4. WeakHashMap(Map m): This constructor is used to create a new WeakHashMap with the same mappings as the specified map.

Methods in WeakHashMap

Method	Action Performed

Method	Action Performed
clear()	Removes all of the mappings from this map. The map will be empty after this call returns.
containsValue(Object value)	Returns true if this map maps one or more keys to the specified value.
containsKey(Object key)	Returns true if this map contains a mapping for the specified key.
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.
get(Object key)	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
isEmpty()	Returns true if this map contains no key-value mappings. This result is a snapshot, and may not reflect unprocessed entries that will be removed before next attempted access because they are no longer referenced.
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.
put(K key, V value)	Associates the specified value with the specified key in this map. If the map previously contained a mapping for this key, the old value is replaced.
putAll(Map 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.

Method	Action Performed
remove(Object key)	Removes the mapping for a key from this weak hash map if it is present. 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.
size()	Returns the number of key-value mappings in this map. This result is a snapshot, and may not reflect unprocessed entries that will be removed before the next attempted access because they are no longer referenced.
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.

Example 1:

```
// Java Program to Illustrate WeakHashMap class
// Via close(), containsValue(), containsKey()
// and isEmpty() method

// Importing required classes
import java.util.Map;
import java.util.WeakHashMap;

// Main class
// WeakHashMapdemo
class GFG {

    // Main driver method
    public static void main(String[] arg)
    {

        // Creating an empty WeakHashMap
        // of Number and string type
        Map<Number, String> weak
            = new WeakHashMap<Number, String>();
```

```
        // Inserting custom elements
        // using put() method
        weak.put(1, "geeks");
        weak.put(2, "for");
        weak.put(3, "geeks");

        // Printing and alongside checking weak map
        System.out.println("our weak map: " + weak);

        // Checking if "for" exist
        if (weak.containsValue("for"))
            System.out.println("for exist");

        // Checking if 1 exist as a key in Map
        if (weak.containsKey(1))
            System.out.println("1 exist");

        // Removing all data
        // using clear() method
        weak.clear();

        // Checking whether the Map is empty or not
        // using isEmpty() method
        if (weak.isEmpty())

            // Display message for better readability
            System.out.println("empty map: " + weak);
    }
}
```

Output

```
our weak map: {3=geeks, 2=for, 1=geeks}
for exist
1 exist
empty map: {}
```

Example 2:

```
// Java Program to Illustrate WeakHashMap class
// Via entrySet(), keySet() and Values() Method
```

```
// Importing required classes
import java.util.Collection;
import java.util.Map;
import java.util.Set;
import java.util.WeakHashMap;

// Main class
// WeakHashMapdemo
class GFG {

    // Main driver method
    public static void main(String[] arg)
    {

        // Creating an empty WeakHashMap
        // of number and string type
        Map<Number, String> weak
            = new WeakHashMap<Number, String>();

        // Inserting elements
        // using put() method
        weak.put(1, "geeks");
        weak.put(2, "for");
        weak.put(3, "geeks");

        // Creating object of Set interface
        Set set1 = weak.entrySet();

        // Checking above Set
        System.out.println(set1);

        // Creating set for key
        Set keySet = weak.keySet();

        // Checking keySet
        System.out.println("key set: " + keySet);

        Collection value = weak.values();

        // Checking values of map and printing them
        System.out.println("values: " + value);
    }
}
```

Output

```
[3=geeks, 2=for, 1=geeks]
key set: [3, 2, 1]
values: [geeks, for, geeks]
```

Example 3:

```
// Java code remove(), putAll()
// get() and size() method

import java.util.Collection;
import java.util.Map;
import java.util.Set;
import java.util.WeakHashMap;

class WeakHashMapdemo {
    public static void main(String[] arg)
    {
        Map<Number, String> weak
            = new WeakHashMap<Number, String>();
        weak.put(1, "geeks");
        weak.put(2, "for");
        weak.put(3, "geeks");

        Map<Number, String> weak1
            = new WeakHashMap<Number, String>();
        weak1.putAll(weak);

        // Getting value of key 2
        System.out.println(weak1.get(2));

        // Printing the size of map
        // using size() method
        System.out.println("Size of map is: "
                           + weak1.size());

        // Removing second element
        // using standard remove() method
        weak1.remove(2);

        // Printing the size after removing key and value
        // pair
        System.out.println("Size after removing: "
                           + weak1.size());
    }
}
```

Output

```
for
Size of map is: 3
Size after removing: 2
```

This article is contributed by **Abhishek Verma**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.