**IdentityHashMap class in Java**

The IdentityHashMap implements [Map](https://www.geeksforgeeks.org/map-interface-java-examples/) interface using [Hashtable](https://www.geeksforgeeks.org/hashtable-in-java/), using reference-equality in place of object-equality when comparing keys (and values). This class is not a general-purpose Map implementation. While this class implements the Map interface, it intentionally violates Map’s general contract, which mandates the use of the equals() method when comparing objects. This class is used when the user requires the objects to be compared via reference. It belongs to java.util package.

Features of IdentityHashMap

Attention reader! Don’t stop learning now. Get hold of all the important [Java Foundation](https://practice.geeksforgeeks.org/courses/Java-Foundation?vC=1) and Collections concepts with the [Fundamentals of Java and Java Collections Course](https://practice.geeksforgeeks.org/courses/Java-Collections?vC=1) at a student-friendly price and become industry ready. To complete your preparation from learning a language to DS Algo and many more,  please refer [Complete Interview Preparation Course](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=GeeksforGeeks&utm_medium=Text&utm_campaign=GFG_Article_Bottom_Text_CIP).

It follows reference equality, instead of using the equals() method it uses the == operator.

It is not synchronized and must be synchronized externally.

Iterators are fail-fast, throw ConcurrentModificationException in an attempt to modify while iterating.

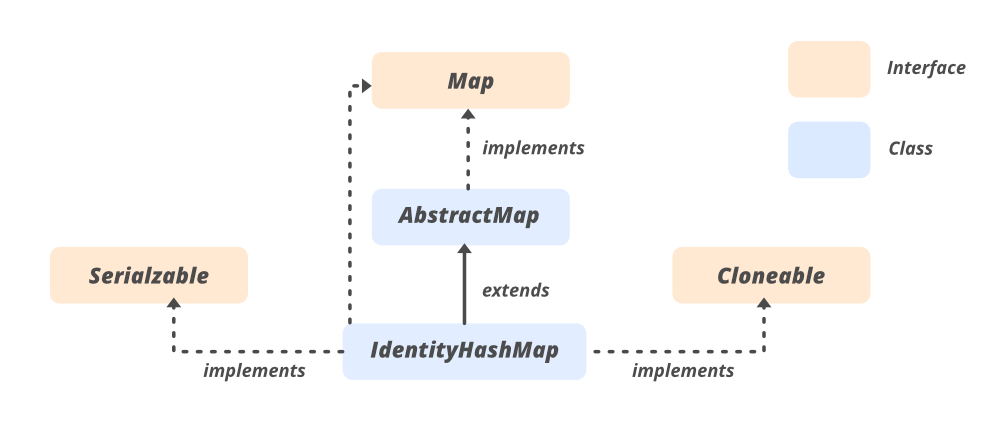
This class provides constant-time performance for the basic operations (get and put), assuming the system identity hash function (System.identityHashCode(Object)) disperses elements properly among the buckets. IdentityHashMap doesn’t use hashCode() method instead it uses System.identityHashCode() method. This is a significant difference because now you can use mutable objects as key in Map whose hash code is likely to change when the mapping is stored inside IdentityHashMap.

Declaration:

public class IdentityHashMap<K,​V> extends AbstractMap<K,​V> implements Map<K,​V>, Serializable, Cloneable 

Here, K is the key Object type and V is the value Object type.

The Hierarchy of IdentityHashMap



It implements Serializable, Cloneable, [Map<K,​ V>](https://www.geeksforgeeks.org/map-interface-java-examples/) interfaces and extends AbstractMap<K, V> class.

Example:

Java

|  |
| --- |
| // Java code to demonstrate IdentityHashMap    import java.util.Map;  import java.util.HashMap;  import java.util.IdentityHashMap;    public class IdentityHashMapExample  {      public static void main(String[] args)      {          // creating an instance of IdentityHashMap          Map<String, String> ihm = new IdentityHashMap<>();            // Putting key and value pair          // in a IdentityHashMap Object          ihm.put("ihmkey","ihmvalue");          ihm.put(new String("ihmkey"),"ihmvalue1");            // ihm.size() will print 2 since it          // compares the objects by reference          System.out.println("Size of IdentityHashMap--"+ihm.size());        }  } |

Output

Size of IdentityHashMap--2

Constructors of IdentityHashMap

We can create an instance of IdentityHashMap in two ways:

IdentityHashMap<K, V> ihm = new IdentityHashMap<K, V>();

(or)

Map<K, V> hm = new IdentityHashMap<K, V>();

1. IdentityHashMap(): Constructs a new, empty identity hash map with a default expected maximum size.

IdentityHashMap<K, V> ihm = new IdentityHashMap<K, V>();

2. IdentityHashMap(int expectedMaxSize): Constructs a new, empty map with the specified expected maximum size.

IdentityHashMap<K, V> ihm = new IdentityHashMap(int expectedMaxSize);

3. IdentityHashMap(Map m): Constructs a new identity hash map containing the key-value mappings in the specified map.

IdentityHashMap<K, V> ihm = new IdentityHashMap(Map m);

Basic Operations on IdentityHashMap

1. Adding Elements

To insert or add mapping into an IdentityHashMap, we have [put()](https://www.geeksforgeeks.org/identityhashmap-put-method-in-java/#:~:text=put()%20method%20of%20IdentityHashMap,replaced%20by%20the%20new%20value.) and [putAll()](https://www.geeksforgeeks.org/identityhashmap-putall-method-in-java/?ref=rp) methods. put() can insert a specific key and the value it is mapping, into a particular map. If an existing key is passed then the previous value gets replaced by the new value. putAll() copies all of the elements i.e., the mappings, from one map into another.

Java

|  |
| --- |
| // Java code to illustrate  // adding elements to IdentityHashMap  import java.util.\*;    public class AddingElementsToIdentityHashMap {        public static void main(String[] args)      {          // Creating an empty IdentityHashMap          Map<Integer, String> identity\_hash              = new IdentityHashMap<Integer, String>();            // Mapping string values to int keys          // using put() method          identity\_hash.put(10, "Geeks");          identity\_hash.put(15, "4");          identity\_hash.put(20, "Geeks");          identity\_hash.put(25, "Welcomes");          identity\_hash.put(30, "You");            // Displaying the IdentityHashMap          System.out.println("Initial Mappings are: "                             + identity\_hash);            // Inserting existing key along with new value            // previous value gets returned and stored in            // returned\_value          String returned\_value              = (String)identity\_hash.put(20, "All");            // Verifying the returned value          System.out.println("Returned value is: "                             + returned\_value);            // Displaying the new map          System.out.println("New map is: " + identity\_hash);            // Creating a new Identityhash map and copying          Map<Integer, String> new\_Identityhash\_map              = new IdentityHashMap<Integer, String>();          new\_Identityhash\_map.putAll(identity\_hash);            // Displaying the final IdentityHashMap          System.out.println("The new map: "                             + new\_Identityhash\_map);      }  } |

Output

Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}

Returned value is: Geeks

New map is: {10=Geeks, 25=Welcomes, 30=You, 20=All, 15=4}

The new map: {10=Geeks, 25=Welcomes, 30=You, 20=All, 15=4}

2. Removing Elements  
To remove mappings, we use [remove()](https://www.geeksforgeeks.org/identityhashmap-remove-method-in-java/), an inbuilt method of IdentityHashMap class, and is used to remove the mapping of any particular key from the map.

Java

|  |
| --- |
| // Java code to illustrate removing  // elements from IdentityHashMap    import java.util.\*;    public class RemovingMappingsFromIdentityHashMap {      public static void main(String[] args)      {            // Creating an empty IdentityHashMap          Map<Integer, String> Identity\_hash = new                      IdentityHashMap<Integer, String>();            // Mapping string values to int keys          Identity\_hash.put(10, "Geeks");          Identity\_hash.put(15, "4");          Identity\_hash.put(20, "Geeks");          Identity\_hash.put(25, "Welcomes");          Identity\_hash.put(30, "You");            // Displaying the IdentityHashMap          System.out.println("Initial Mappings are: " +                                          Identity\_hash);            // Removing the existing key mapping          String returned\_value =                          (String)Identity\_hash.remove(20);            // Verifying the returned value          System.out.println("Returned value is: " +                                      returned\_value);            // Displaying the new map          System.out.println("New map is: " + Identity\_hash);      }  } |

Output

Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}

Returned value is: Geeks

New map is: {10=Geeks, 25=Welcomes, 30=You, 15=4}

3. Accessing the Elements

We can access the elements of an IdentityHashMap using the [get()](https://www.geeksforgeeks.org/identityhashmap-get-method-in-java/) method, the example of this is given below.

Java

|  |
| --- |
| // Java code to illustrate the accessing  // elements from IdentityHashMap    import java.util.\*;    public class AccessingElementsFromIdentityHashMap {        public static void main(String[] args)      {            // Creating an empty IdentityHashMap          Map<Integer, String> identity\_hash              = new IdentityHashMap<Integer, String>();            // Mapping string values to int keys          identity\_hash.put(10, "Geeks");          identity\_hash.put(15, "4");          identity\_hash.put(20, "Geeks");          identity\_hash.put(25, "Welcomes");          identity\_hash.put(30, "You");            // Displaying the IdentityHashMap          System.out.println("Initial Mappings are: "                             + identity\_hash);            // Getting the value of 25          System.out.println("The Value is: "                             + identity\_hash.get(25));            // Getting the value of 10          System.out.println("The Value is: "                             + identity\_hash.get(10));              // Using keySet() to get the set view of keys          System.out.println("The set is: " + identity\_hash.keySet());              // Using entrySet() to get the set view          System.out.println("The set is: " +                                  identity\_hash.entrySet());      }  } |

Output

Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}

The Value is: Welcomes

The Value is: Geeks

The set is: [10, 25, 30, 20, 15]

The set is: [10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4]

4. Traversing  
We can use the Iterator interface to traverse over any structure of the Collection Framework. Since Iterators work with one type of data we use Entry< ? , ? > to resolve the two separate types into a compatible format. Then using the next() method we print the elements of the IdentityHashMap.

Java

|  |
| --- |
| // Java code to illustrate the  // iterating over IdentityHashmap    import java.util.\*;    public class IteratingIdentityHashMap {        public static void main(String[] args)      {            // Creating an empty IdentityHashMap          IdentityHashMap<Integer, String> identity\_hash              = new IdentityHashMap<Integer, String>();            // Mapping string values to int keys          identity\_hash.put(10, "Geeks");          identity\_hash.put(15, "4");          identity\_hash.put(20, "Geeks");          identity\_hash.put(25, "Welcomes");          identity\_hash.put(30, "You");            // Displaying the IdentityHashMap          System.out.println("Initial Mappings are: "                             + identity\_hash);            // Create an Iterator over the          // IdentityHashMap          Iterator<IdentityHashMap.Entry<Integer, String> >              itr = identity\_hash.entrySet().iterator();            // The hasNext() method is used to check if there is          // a next element The next() method is used to          // retrieve the next element          while (itr.hasNext()) {              IdentityHashMap.Entry<Integer, String> entry                  = itr.next();              System.out.println("Key = " + entry.getKey()                                 + ", Value = "                                 + entry.getValue());          }      }  } |

Output

Initial Mappings are: {10=Geeks, 25=Welcomes, 30=You, 20=Geeks, 15=4}

Key = 10, Value = Geeks

Key = 25, Value = Welcomes

Key = 30, Value = You

Key = 20, Value = Geeks

Key = 15, Value = 4

Synchronized IdentityHashMap

If multiple threads access an identity 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 IdentityHashMap(…));

Methods of IdentityHashMap

K – The type of the keys in the map.

V – The type of values mapped in the map.

| METHOD | DESCRIPTION |
| --- | --- |
| [clear()](https://www.geeksforgeeks.org/identityhashmap-clear-method-in-java/) | Removes all of the mappings from this map. |
| [clone()](https://www.geeksforgeeks.org/identityhashmap-clone-method-in-java/) | Returns a shallow copy of this identity hash map: the keys and values themselves are not cloned. |
| [containsKey​(Object key)](https://www.geeksforgeeks.org/identityhashmap-containskey-method-in-java/) | Tests whether the specified object reference is a key in this identity hash map. |
| [containsValue​(Object value)](https://www.geeksforgeeks.org/identityhashmap-containsvalue-method-in-java/) | Tests whether the specified object reference is a value in this identity hash map. |
| [entrySet()](https://www.geeksforgeeks.org/identityhashmap-entryset-method-in-java/) | Returns a [Set](https://www.geeksforgeeks.org/set-in-java/) view of the mappings contained in this map. |
| [equals​(Object o)](https://www.geeksforgeeks.org/identityhashmap-equals-method-in-java/) | Compares the specified object with this map for equality. |
| [get​(Object key)](https://www.geeksforgeeks.org/identityhashmap-get-method-in-java/) | Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| [hashCode()](https://www.geeksforgeeks.org/identityhashmap-hashcode-method-in-java/) | Returns the hash code value for this map. |
| [isEmpty()](https://www.geeksforgeeks.org/identityhashmap-isempty-method-in-java/) | Returns true if this identity hash map contains no key-value mappings. |
| [keySet()](https://www.geeksforgeeks.org/identityhashmap-keyset-method-in-java/) | Returns an identity-based set view of the keys contained in this map. |
| [put​(K key, V value)](https://www.geeksforgeeks.org/identityhashmap-put-method-in-java/) | Associates the specified value with the specified key in this identity hash map. |
| [putAll​(Map<? extends K,​? extends V> m)](https://www.geeksforgeeks.org/identityhashmap-putall-method-in-java/) | Copies all of the mappings from the specified map to this map. |
| [remove​(Object key)](https://www.geeksforgeeks.org/identityhashmap-remove-method-in-java/) | Removes the mapping for this key from this map if present. |
| [size()](https://www.geeksforgeeks.org/identityhashmap-size-method-in-java/) | Returns the number of key-value mappings in this identity hash map. |
| [values()](https://www.geeksforgeeks.org/identityhashmap-values-method-in-java/) | Returns a Collection view of the values contained in this map. |

Methods declared in class java.util.AbstractMap

| METHOD | DESCRIPTION |
| --- | --- |
| toString() | Returns a string representation of this map. |

Methods declared in interface java.util.Map

| METHOD | DESCRIPTION |
| --- | --- |
| 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). |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| 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. |
| remove​(Object key, Object value) | Removes the entry for the specified key only if it is currently mapped to the specified value. |
| replace​(K key, V value) | Replaces the entry for the specified key only if it is currently mapped to some value. |
| replace​(K key, V oldValue, V newValue) | Replaces the entry for the specified key only if currently mapped to the specified value. |
| 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. |

IdentityHashMap vs [HashMap](https://www.geeksforgeeks.org/hashmap-treemap-java/)

IdentityHashMap uses the equality operator “==” for comparing keys and values while HashMap uses the equals method for comparing keys and values inside Map.

Since IdentityHashMap doesn’t use equals() its comparatively faster than HashMap for an object with expensive equals().

IdentityHashMap doesn’t require keys to be immutable as it is not relied on equals().

The below program illustrates the difference between IdentityHashMap and HashMap implementation.

Java

|  |
| --- |
| // Java code to demonstrate IdentityHashMap and  // illustration of how it is different from HashMap    import java.util.Map;  import java.util.HashMap;  import java.util.IdentityHashMap;    public class IdentityHashMapExample  {      public static void main(String[] args)      {          // Creating HashMap and IdentityHashMap objects          Map<String, String> hm = new HashMap<>();          Map<String, String> ihm = new IdentityHashMap<>();            // Putting key and value in HashMap and IdentityHashMap Object          hm.put("hmkey","hmvalue");          hm.put(new String("hmkey"),"hmvalue1");          ihm.put("ihmkey","ihmvalue");          ihm.put(new String("ihmkey"),"ihmvalue1");            // Print Size of HashMap and WeakHashMap Object          // hm.size() will print 1 since it compares the objects logically          // and both the keys are same          System.out.println("Size of HashMap is : "+hm.size());            // ihm.size() will print 2 since it compares the objects by reference          System.out.println("Size of IdentityHashMap is : "+ihm.size());          }  } |

Output

Size of HashMap is : 1

Size of IdentityHashMap is : 2

**WeakHashMap class in Java**

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. Few important features of a weak hashmap are:

Both null values and null keys are supported in WeakHashMap.

It is not synchronised.

This class is intended primarily for use with key objects whose equals methods test for object identity using the == operator.

Constructors in WeakHashMap:

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

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

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

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

Methods in WeakHashMap:

[void clear():](https://www.geeksforgeeks.org/weakhashmap-clear-method-in-java/) The method removes all of the mappings from this map. The map will be empty after this call returns.

Syntax: public void clear().

Returns: NA.

Exception: NA.

[boolean containsValue(Object value):](https://www.geeksforgeeks.org/weakhashmap-containsvalue-method-in-java/) this method returns true if this map maps one or more keys to the specified value.

Syntax: public boolean containsValue(Object value).

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

Exception: NA

[boolean containsKey(Object key):](https://www.geeksforgeeks.org/weakhashmap-containskey-method-in-java/) This method returns true if this map contains a mapping for the specified key.

Syntax: public boolean containsKey(Object key).

Returns: true if there is a mapping for key; false otherwise.

Exception: NA

[put(K key, V value):](https://www.geeksforgeeks.org/weakhashmap-put-method-in-java/) 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.

Syntax: public put(K key, V value).

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.)

Exception: NA

[boolean isEmpty():](https://www.geeksforgeeks.org/weakhashmap-isempty-method-in-java-with-examples/) 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.

Syntax: public boolean isEmpty()

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

Exceptions: NA

|  |
| --- |
| // Java code illustrating close(), containsValue()  // containsKey() and isEmpty() method    import java.util.Map;  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");            // 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          weak.clear();            // Checking whether map is empty or not          if(weak.isEmpty())              System.out.println("empty map: " + weak);      }  } |

Output:

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

for exist

1 exist

empty map: {}

[Set entrySet():](https://www.geeksforgeeks.org/weakhashmap-entryset-method-in-java/) 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.

Syntax: public Set entrySet()

Returns: a set view of the mappings contained in this map.

Exception: NA

[Set keySet():](https://www.geeksforgeeks.org/weakhashmap-keyset-method-in-java/) 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.

Syntax: public Set keySet().

Returns: a set view of the keys contained in this map

Exception: NA

[Collection values():](https://www.geeksforgeeks.org/weakhashmap-values-method-in-java/) 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.

Syntax: public Collection values().

Returns: a collection view of the values contained in this map

Exception: NA

|  |
| --- |
| // Java code illustrating entrySet(), keySet() and Values()    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");            Set set1 = weak.entrySet();            // Checking 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          System.out.println("values: " + value);      }  } |

Output:

[3=geeks, 2=for, 1=geeks]

key set: [3, 2, 1]

values: [geeks, for, geeks]

[void putAll(Map m):](https://www.geeksforgeeks.org/weakhashmap-putall-method-in-java/) 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.

Syntax: public void putAll(Map m)

Returns: NA

Exception:

NullPointerException - if the specified map is null.

[V get(Object key):](https://www.geeksforgeeks.org/weakhashmap-get-method-in-java/) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

Syntax: 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

Exception: NA

[V remove(Object key):](https://www.geeksforgeeks.org/weakhashmap-remove-method-in-java/) 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.

Syntax: public V remove(Object key)

Returns: the previous value associated with key, or

null if there was no mapping for key

Exception: NA.

[int size():](https://www.geeksforgeeks.org/weakhashmap-size-method-in-java/) 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 next attempted access because they are no longer referenced.

Syntax: public int size()

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

Exception: NA

|  |
| --- |
| // 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));            // size of map          System.out.println("Size of map is: " + weak1.size());            // removing 2nd element          weak1.remove(2);            // 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