### Methods of Java HashMap class

| **Method** | **Description** |
| --- | --- |
| void clear() | It is used to remove all of the mappings from this map. |
| boolean isEmpty() | It is used to return true if this map contains no key-value mappings. |
| Object clone() | It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned. |
| Set entrySet() | It is used to return a collection view of the mappings contained in this map. |
| Set keySet() | It is used to return a set view of the keys contained in this map. |
| V put(Object key, Object value) | It is used to insert an entry in the map. |
| void putAll(Map map) | It is used to insert the specified map in the map. |
| V putIfAbsent(K key, V value) | It inserts the specified value with the specified key in the map only if it is not already specified. |
| V remove(Object key) | It is used to delete an entry for the specified key. |
| boolean remove(Object key, Object value) | It removes the specified values with the associated specified keys from the map. |
| V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) | It is used 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) | It is used to compute its value using the given mapping function, if the specified key is not already associated with a value (or is mapped to null), and enters it into this map unless null. |
| V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) | It is used to compute a new mapping given the key and its current mapped value if the value for the specified key is present and non-null. |
| boolean containsValue(Object value) | This method returns true if some value equal to the value exists within the map, else return false. |
| boolean containsKey(Object key) | This method returns true if some key equal to the key exists within the map, else return false. |
| boolean equals(Object o) | It is used to compare the specified Object with the Map. |
| void forEach(BiConsumer<? super K,? super V> action) | It performs the given action for each entry in the map until all entries have been processed or the action throws an exception. |
| V get(Object key) | This method returns the object that contains the value associated with the key. |
| V getOrDefault(Object key, V defaultValue) | It returns the value to which the specified key is mapped, or defaultValue if the map contains no mapping for the key. |
| boolean isEmpty() | This method returns true if the map is empty; returns false if it contains at least one key. |
| 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 replace(K key, V value) | It replaces the specified value for a specified key. |
| boolean replace(K key, V oldValue, V newValue) | It replaces the old value with the new value for a specified key. |
| void replaceAll(BiFunction<? super K,? super V,? extends V> function) | It 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. |
| Collection<V> values() | It returns a collection view of the values contained in the map. |
| int size() | This method returns the number of entries in the map. |

## **Other Methods of HashMap**

| Method | Description |
| --- | --- |
| [clear()](https://www.programiz.com/java-programming/library/hashmap/clear) | removes all mappings from the HashMap |
| [compute()](https://www.programiz.com/java-programming/library/hashmap/compute) | computes a new value for the specified key |
| [computeIfAbsent()](https://www.programiz.com/java-programming/library/hashmap/computeifabsent) | computes value if a mapping for the key is not present |
| [computeIfPresent()](https://www.programiz.com/java-programming/library/hashmap/computeifpresent) | computes a value for mapping if the key is present |
| [merge()](https://www.programiz.com/java-programming/library/hashmap/merge) | merges the specified mapping to the HashMap |
| [clone()](https://www.programiz.com/java-programming/library/hashmap/clone) | makes the copy of the HashMap |
| [containsKey()](https://www.programiz.com/java-programming/library/hashmap/containskey) | checks if the specified key is present in Hashmap |
| [containsValue()](https://www.programiz.com/java-programming/library/hashmap/containsvalue) | checks if Hashmap contains the specified value |
| [size()](https://www.programiz.com/java-programming/library/hashmap/size) | returns the number of items in HashMap |
| [isEmpty()](https://www.programiz.com/java-programming/library/hashmap/isempty) | checks if the Hashmap is empty |

## **Iterate through a HashMap**

To iterate through each entry of the hashmap, we can use [Java for-each loop](https://www.programiz.com/java-programming/enhanced-for-loop). We can iterate through keys only, vales only, and key/value mapping. For example,

import java.util.HashMap;

import java.util.Map.Entry;

class Main {

public static void main(String[] args) {

// create a HashMap

HashMap<Integer, String> languages = new HashMap<>();

languages.put(1, "Java");

languages.put(2, "Python");

languages.put(3, "JavaScript");

System.out.println("HashMap: " + languages);

// iterate through keys only

System.out.print("Keys: ");

for (Integer key : languages.keySet()) {

System.out.print(key);

System.out.print(", ");

}

// iterate through values only

System.out.print("\nValues: ");

for (String value : languages.values()) {

System.out.print(value);

System.out.print(", ");

}

// iterate through key/value entries

System.out.print("\nEntries: ");

for (Entry<Integer, String> entry : languages.entrySet()) {

System.out.print(entry);

System.out.print(", ");

}

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Output

HashMap: {1=Java, 2=Python, 3=JavaScript}

Keys: 1, 2, 3,

Values: Java, Python, JavaScript,

Entries: 1=Java, 2=Python, 3=JavaScript,

Note that we have used the Map.Entry in the above example. It is the nested class of the Map interface that returns a view (elements) of the map.

We first need to import the java.util.Map.Entry package in order to use this class.

This nested class returns a view (elements) of the map.

## **Creating HashMap from Other Maps**

In Java, we can also create a hashmap from other maps. For example,

import java.util.HashMap;

import java.util.TreeMap;

class Main {

public static void main(String[] args) {

// create a treemap

TreeMap<String, Integer> evenNumbers = new TreeMap<>();

evenNumbers.put("Two", 2);

evenNumbers.put("Four", 4);

System.out.println("TreeMap: " + evenNumbers);

// create hashmap from the treemap

HashMap<String, Integer> numbers = new HashMap<>(evenNumbers);

numbers.put("Three", 3);

System.out.println("HashMap: " + numbers);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

Output

TreeMap: {Four=4, Two=2}

HashMap: {Two=2, Three=3, Four=4}

In the above example, we have created a TreeMap named evenNumbers. Notice the expression,

numbers = new HashMap<>(evenNumbers)

Here, we are creating a HashMap named numbers using the TreeMap. To learn more about treemap, visit [Java TreeMap](https://www.programiz.com/java-programming/treemap).

Note: While creating a hashmap, we can include optional parameters: capacity and load factor. For example,

HashMap<K, V> numbers = new HashMap<>(8, 0.6f);

Here,

* 8 (capacity is 8) - This means it can store 8 entries.
* 0.6f (load factor is 0.6) - This means whenever our hash table is filled by 60%, the entries are moved to a new hash table double the size of the original hash table.

If the optional parameters not used, then the default capacity will be 16 and the default load factor will be 0.75.