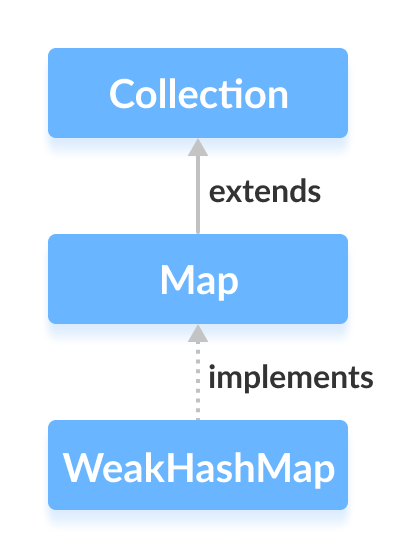
**Java WeakHashMap**

The WeakHashMap class of the Java collections framework provides the feature of the hash table data structure..

It implements the Map interface.



**Note**: Keys of the weak hashmap are of the **WeakReference** type.

The object of a weak reference type can be garbage collected in Java if the reference is no longer used in the program.

Let us learn to create a weak hash map first. Then, we will learn how it differs from a hashmap.

## Create a WeakHashMap

In order to create a weak hashmap, we must import the java.util.WeakHashMap package first. Once we import the package, here is how we can create weak hashmaps in Java.

//WeakHashMap creation with capacity 8 and load factor 0.6

WeakHashMap<Key, Value> numbers = new WeakHashMap<>(8, 0.6);

In the above code, we have created a weak hashmap named numbers.

Here,

* Key - a unique identifier used to associate each element (value) in a map
* Value - elements associated by keys in a map

Notice the part new WeakHashMap<>(8, 0.6). Here, the first parameter is **capacity** and the second parameter is **loadFactor**.

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

**Default capacity and load factor**

It is possible to create a weak hashmap without defining its capacity and load factor. For example,

// WeakHashMap with default capacity and load factor

WeakHashMap<Key, Value> numbers1 = new WeakHashMap<>();

By default,

* the capacity of the map will be 16
* the load factor will be 0.75

## Differences Between HashMap and WeakHashMap

Let us see the implementation of a weak hashmap in Java.

import java.util.WeakHashMap;

class Main {

public static void main(String[] args) {

// Creating WeakHashMap of numbers

WeakHashMap<String, Integer> numbers = new WeakHashMap<>();

String two = new String("Two");

Integer twoValue = 2;

String four = new String("Four");

Integer fourValue = 4;

// Inserting elements

numbers.put(two, twoValue);

numbers.put(four, fourValue);

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

// Make the reference null

two = null;

// Perform garbage collection (waise to automatic)

System.gc();

System.out.println("WeakHashMap after garbage collection: " + numbers);

}

}

**Output**

WeakHashMap: {Four=4, Two=2}

WeakHashMap after garbage collection: {Four}

As we can see, when the key two of a weak hashmap is set to null and perform garbage collection, the key is removed.

It is because unlike keys of weak hashmaps are of **weak reference** type. This means the entry of a map are removed by the garbage collector if the key to that entry is no longer used. This is useful to save resources.

Now let us see the same implementation in a hashmap.

import java.util.HashMap;

class Main {

public static void main(String[] args) {

// Creating HashMap of even numbers

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

String two = new String("Two");

Integer twoValue = 2;

String four = new String("Four");

Integer fourValue = 4;

// Inserting elements

numbers.put(two, twoValue);

numbers.put(four, fourValue);

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

// Make the reference null

two = null;

// Perform garbage collection

System.gc();

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

}

}

**Output**

HashMap: {Four=4, Two=2}

HashMap after garbage collection: {Four=4, Two=2}

Here, when the key two of the hashmap is set to null and perform garbage collection, the key is not removed.

This is because unlike weak hashmaps keys of hashmaps are of **strong reference** type. This means the entry of a map is not removed by the garbage collector even though the key to that entry is no longer used.

**Note**: **All functionalities of hashmaps and weak hashmaps are similar except keys of a weak hashmap are of weak reference, whereas keys of a hashmap are of strong reference.**

## Creating WeakHashMap from Other Maps

Here is how we can create a weak hashmap from other maps.

import java.util.HashMap;

import java.util.WeakHashMap;

class Main {

public static void main(String[] args) {

// Creating a hashmap of even numbers

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

String two = new String("Two");

Integer twoValue = 2;

evenNumbers.put(two, twoValue);

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

// Creating a weak hash map from other hashmap

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

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

}

}

**Output**

HashMap: {Two=2}

WeakHashMap: {Two=2}

## Methods of WeakHashMap

The WeakHashMap class provides methods that allow us to perform various operations on the map.

## Insert Elements to WeakHashMap

* put() - inserts the specified key/value mapping to the map
* putAll() - inserts all the entries from specified map to this map
* putIfAbsent() - inserts the specified key/value mapping to the map if the specified key is not present in the map

For example,

import java.util.WeakHashMap;

class Main {

public static void main(String[] args) {

// Creating WeakHashMap of even numbers

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

String two = new String("Two");

Integer twoValue = 2;

// Using put()

evenNumbers.put(two, twoValue);

String four = new String("Four");

Integer fourValue = 4;

// Using putIfAbsent()

evenNumbers.putIfAbsent(four, fourValue);

System.out.println("WeakHashMap of even numbers: " + evenNumbers);

//Creating WeakHashMap of numbers

WeakHashMap<String, Integer> numbers = new WeakHashMap<>();

String one = new String("One");

Integer oneValue = 1;

numbers.put(one, oneValue);

// Using putAll()

numbers.putAll(evenNumbers);

System.out.println("WeakHashMap of numbers: " + numbers);

}

}

**Output**

WeakHashMap of even numbers: {Four=4, Two=2}

WeakHashMap of numbers: {Two=2, Four=4, One=1}

## Access WeakHashMap Elements

**1. Using entrySet(), keySet() and values()**

* entrySet() - returns a set of all the key/value mapping of the map
* keySet() - returns a set of all the keys of the map
* values() - returns a set of all the values of the map

For example,

import java.util.WeakHashMap;

class Main {

public static void main(String[] args) {

// Creating WeakHashMap of even numbers

WeakHashMap<String, Integer> numbers = new WeakHashMap<>();

String one = new String("One");

Integer oneValue = 1;

numbers.put(one, oneValue);

String two = new String("Two");

Integer twoValue = 2;

numbers.put(two, twoValue);

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

// Using entrySet()

System.out.println("Key/Value mappings: " + numbers.entrySet());

// Using keySet()

System.out.println("Keys: " + numbers.keySet());

// Using values()

System.out.println("Values: " + numbers.values());

}

}

**Output**

WeakHashMap: {Two=2, One=1}

Key/Value mappings: [Two=2, One=1]

Keys: [Two, One]

Values: [1, 2]

**2. Using get() and getOrDefault()**

* get() - Returns the value associated with the specified key. Returns null if the key is not found.
* getOrDefault() - Returns the value associated with the specified key. Returns the specified default value if the key is not found.

For example,

import java.util.WeakHashMap;

class Main {

public static void main(String[] args) {

// Creating WeakHashMap of even numbers

WeakHashMap<String, Integer> numbers = new WeakHashMap<>();

String one = new String("One");

Integer oneValue = 1;

numbers.put(one, oneValue);

String two = new String("Two");

Integer twoValue = 2;

numbers.put(two, twoValue);

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

// Using get()

int value1 = numbers.get("Two");

System.out.println("Using get(): " + value1);

// Using getOrDefault()

int value2 = numbers.getOrDefault("Four", 4);

System.out.println("Using getOrDefault(): " + value2);

}

}

**Output**

WeakHashMap: {Two=2, One=1}

Using get(): 2

Using getOrDefault(): 4

## Remove WeakHashMap Elements

* remove(key) - returns and removes the entry associated with the specified key from the map
* remove(key, value) - removes the entry from the map only if the specified key mapped to the specified value and return a boolean value

For example,

import java.util.WeakHashMap;

class Main {

public static void main(String[] args) {

// Creating WeakHashMap of even numbers

WeakHashMap<String, Integer> numbers = new WeakHashMap<>();

String one = new String("One");

Integer oneValue = 1;

numbers.put(one, oneValue);

String two = new String("Two");

Integer twoValue = 2;

numbers.put(two, twoValue);

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

// Using remove() with single parameter

int value = numbers.remove("Two");

System.out.println("Removed value: " + value);

// Using remove() with 2 parameters

boolean result = numbers.remove("One", 3);

System.out.println("Is the entry {One=3} removed? " + result);

System.out.println("Updated WeakHashMap: " + numbers);

}

}

**Output**

WeakHashMap: {Two=2, One=1}

Removed value: 2

Is the entry {One=3} removed? False

Updated WeakHashMap: {One=1}

## Other Methods of WeakHashMap

|  |  |
| --- | --- |
| Method | Description |
| clear() | Removes all the entries from the map |
| containsKey() | Checks if the map contains the specified key and returns a boolean value |
| containsValue() | Checks if the map contains the specified value and returns a boolean value |
| size() | Returns the size of the map |
| isEmpty() | Checks if the map is empty and returns a boolean value |

To learn more, visit Java WeakHashMap official documentation.