HowToDoInJava

How to compare two hashmaps in Java

Last Updated: June 27, 2020 By: Lokesh Gupta Java HashMap Java Map

Learn to **compare two hashmaps** in Java by keys, values and key-value pairs. Also learn to compare while allowing or restricting duplicate values.

1. Compare hashmap for same key-values – HashMap.equals()

By default, **HashMap.equals()** method compares two hashmaps by key-value pairs. It means both hashmap instances must have **exactly same key-value pairs** and both must be of same size.

The order of key-value pairs can be different and does not play in role in comparison.

```
Compare hashmaps by key-value pairs

import java.util.HashMap;

public class HashMapExample
{
    public static void main(String[] args) throws CloneNotSupportedException
    {
        HashMap<Integer, String> map1 = new HashMap<>();

        map1.put(1, "A");
        map1.put(2, "B");
        map1.put(3, "C");

        //Same as map1
        HashMap<Integer, String> map2 = new HashMap<>();

    map2.put(3, "C");
        map2.put(1, "A");
        map2.put(2, "B");

        //Different from map1
        HashMap<Integer, String> map3 = new HashMap<>();

        map3.put(1, "A");
        map3.put(2, "B");
        map3.put(3, "C");
        map3.put(3, "D");

        System.out.println(map1.equals(map2));  //true
        System.out.println(map1.equals(map3));  //false
    }
}
```

Program output.

```
Console

true
false
```

2. Compare two hashmaps for same keys – HashMap.keySet()

2.1. Are both hashmaps equal?

If we want to **compare hashmaps by keys** i.e. two hashmaps will be equals if they have **exactly same set of keys**, we can use **HashMap.keySet()** function. It returns all the map keys in **HashSet**.

We can compare the hashset of keys for both maps using **Set.equals()** method. It returns `true` if the two sets have the same size, and every element of the specified set is contained in another set.

```
Compare hashmaps by key set

import java.util.HashMap;

public class HashMapExample
{
    public static void main(String[] args) throws CloneNotSupportedException
    {
        HashMap<Integer, String> map1 = new HashMap<>();

        map1.put(1, "A");
        map1.put(2, "B");
        map1.put(3, "C");

        //Same keys as map1
        HashMap<Integer, String> map2 = new HashMap<>();

    map2.put(3, "C");
        map2.put(1, "A");
        map2.put(2, "B");

        //Different keys than map1
        HashMap<Integer, String> map3 = new HashMap<>();

        map3.put(1, "A");
        map3.put(2, "B");
        map3.put(3, "C");
        map3.put(3, "D");

        System.out.println( map1.keySet().equals( map2.keySet() ));  //true
        System.out.println( map1.keySet().equals( map3.keySet() ));  //false
    }
}
```

Program output.

```
Console

true
false
```

2.2. Find out extra keys

We may be interested in finding out what extra keys first hashmap has than second hashmap. So get this difference, first do a union of keys from both hashmaps, and then remove all keys present in first hashmap.

Java program to find out the **difference between two hashmaps**.

```
Compare hashmaps to find out extra keys

//map 1 has 3 keys
HashMap<Integer, String> map1 = new HashMap<>();

map1.put(1, "A");
map1.put(2, "B");
map1.put(3, "C");

//map 2 has 4 keys
HashMap<Integer, String> map2 = new HashMap<>();

map2.put(1, "A");
map2.put(2, "B");
map2.put(3, "C");
map2.put(4, "C");

//Union of keys from both maps
HashSet<Integer> unionKeys = new HashSet<>(map1.keySet());
unionKeys.addAll(map2.keySet());

unionKeys.removeAll(map1.keySet());

System.out.println(unionKeys);
```

Program output.

```
Console

[4]
```

3. Compare hashmaps for values – HashMap.values()

If we want to compare hashmaps by values i.e. two hashmaps will be equals if they have **exactly same set of values**. Please note that HashMap allows duplicate values, so decide if you want to compare hashmaps **with duplicate or without duplicate values**.

3.1. Duplicates are NOT allowed

Add all values from `HashMap.values()` to an arraylist for both maps. Now compare both arraylists for equality.

```
Compare hashmaps for distinct values

HashMap<Integer, String> map1 = new HashMap<>();

map1.put(1, "A");
map1.put(2, "B");
map1.put(3, "C");

//Same values as map1
HashMap<Integer, String> map2 = new HashMap<>();
```

```
map2.put(4, "A");
map2.put(5, "B");
map2.put(6, "C");

//Different values than map1 - C is added twice
HashMap<Integer, String> map3 = new HashMap<>();

map3.put(1, "A");
map3.put(2, "B");
map3.put(3, "C");
map3.put(4, "C");

System.out.println( new ArrayList<>( map1.values() ).equals(new ArrayList<>( map2.values() )) );       //true
System.out.println( new ArrayList<>( map1.values() ).equals(new ArrayList<>( map3.values() )) );       //false
```

Program output.

```
Console

true
false
```

3.2. Duplicates are allowed

If you want to remove duplicate values before comparing the hashmaps, the add all values into a **HashSet** which automatically ignores duplicate values.

```
Compare hashmaps for distinct values

HashMap<Integer, String> map1 = new HashMap<>();

map1.put(1, "A");
map1.put(2, "B");
map1.put(3, "C");

//Same values as map1
HashMap<Integer, String> map2 = new HashMap<>();

map2.put(4, "A");
map2.put(5, "B");
map2.put(6, "C");

//Duplicate values  - C is added twice
HashMap<Integer, String> map3 = new HashMap<>();

map3.put(1, "A");
map3.put(2, "B");
map3.put(3, "C");
map3.put(4, "C");

System.out.println( new HashSet<>( map1.values() ).equals(new HashSet<>( map2.values() )) );       //true
System.out.println( new HashSet<>( map1.values() ).equals(new HashSet<>( map3.values() )) );       //true
```

Program output.

```
Console

true
true
```