Java - HashMap confusion about collision handling and the get() method

Ask Question

I'm using a `HashMap` and I haven't been able to get a straight answer on how the `get()` method works in the case of collisions.

Let's say `n > 1` objects get placed in the same **key**. Are they stored in a `LinkedList` ? Are they overwritten so that only the last object placed in that key exists there anymore? Are they using some other collision method?

If they are placed in a `LinkedList` , is there a way to retrieve that entire list? If not, is there some other built in map for **Java** in which I can do this?

For my purposes, separate chaining would be ideal, as if there are collisions, I need to be able to look through the list and get information about all the objects in it. What would be the best way to do this in **Java**?

Thanks for all your help!

java hashmap

edited Oct 18 '12 at 1:48
Russell Gutierrez
1,250 6 17

asked Oct 18 '12 at 1:39
Slims
437 6 22

1 I think you are a bit confused on the idea of collisions. A collision happens when distinct keys produce the same `hashCode()` value. In this case, typically there are *buckets* at the mapped index. – Chris Dargis Oct 18 '12 at 1:51

Hm yes, I seem to have confused myself. What about when the same key gets used later for a different object? Isn't that a collision since the key will produce the same hash code value? – Slims Oct 18 '12 at 1:53

If it is the same key (the key's `equals()` method returns true) AND you are adding to the map (see @Jigar Joshi 's answer), then it is overwritten. Key != hash code value. – Chris Dargis Oct 18 '12 at 1:55

5 Answers

> Are they overwritten so that only the last object placed in that key exists there anymore?

Yes, assuming you're putting multiple values with the same key (according to `Object.equals`, not `Object.hashCode`.) That's specified in the `Map.put` javadoc:

> If the map previously contained a mapping for the key, the old value is replaced by the specified value.

If you want to map a key to multiple values, you're probably better off using something like Guava's `ListMultimap`, `ArrayListMultimap` in specific, which maps keys to lists of values. (Disclosure: I contribute to Guava.) If you can't tolerate a third-party library, then really you have to have a `Map<Key, List<Value>>`, though that can get a bit unwieldy.

answered Oct 18 '12 at 1:43

Louis Wasserman
139k 17 239 312

Thank you. I'm annoyed that I am going to have to use that latter strategy, but such is life. – Slims Oct 18 '12 at 1:49

The Java designers decided that the "overwriting" behavior was much more commonly useful, and could be used to simulate the "mapping to lists" (with, of course, a `Map<Key, List<Value>>`) -- and that mapping to lists wasn't common enough a use case to add to the JDK itself. – Louis Wasserman Oct 18 '12 at 2:00

The documentation for Hashmap.put() clearly states, "Associates the specified value with the specified key in this map. **If the map previously contained a mapping for the key, the old value is replaced**"

If you would like to have a list of objects associated with a key, then store a list as the value.

Note that 'collision' generally refers to the internal working of the HashMap, where two keys have the same hash value, not the use of the same key for two different values.

edited Oct 18 '12 at 2:15

answered Oct 18 '12 at 1:43

GreyBeardedGeek

19.1k 1 24 41

The OP is referring to collisions resulting from multiple `hashCode()` invocations from two distinct objects producing the same value. Collisions – Chris Dargis Oct 18 '12 at 1:46

1 No, clearly OP was asking about using the same key multiple times, "Let's say n > 1 objects get placed in the same key"... – GreyBeardedGeek Oct 18 '12 at 1:54

Good point, this is clearly stated in the question. If you do a quick edit I will remove my down-vote. Although I still think the OP was referring to collisions, it isn't clear at all. – Chris Dargis Oct 18 '12 at 2:04

@DougRamsey - what did you want me to edit? – GreyBeardedGeek Oct 18 '12 at 2:07

There is nothing wrong with the answer. For me to be allowed to remove the down-vote, an edit needs to be applied to it. – Chris Dargis Oct 18 '12 at 2:09

> Let's say n > 1 objects get placed in the same key. Are they stored in a linked list? Are they overwritten so that only the last object placed in that key exists there anymore? Are they using some other collision method?

There could be single instance for the same key so the last one overrides the prior one

```
Map<String, Integer> map = new HashMap<String, Integer>();
map.put("a", 1);
map.put("a", 2);// it overrides 1 and puts 2 there
```

chaining comes where there turns the same hash for different keys

See

- Java papers hash table working

answered Oct 18 '12 at 1:43

Jigar Joshi

191k 32 326 377

Cite: "Let's say n > 1 objects get placed in the same key. Are they stored in a linked list? Are they overwritten so that only the last object placed in that key exists there anymore? Are they using some other collision method?"

Yes, if the hashmap contained something under this key, it will override it.

You can implement your own class to handle that or more simple use a HashMap> in where K is your Key Object and V the object value. Have in mind that with last solution when you do a map.get(K) will retrieve a List or the implementation that you choose (i.e: ArrayList) so all the methods of this implementation are available for you and perhaps fulfils your requirements. For example if you used Arraylist you have the size, trimToSize, removeRange, etc.

answered Oct 18 '12 at 2:02

sgroh
296 1 12

collision resolution for hashing in java is not based on chaining. To my understanding, JDK uses double hashing which is one of the best way of open addressing. So there's no list going to be associated with a hash slot.

You might put the objects for which the hash function resolves to the same key can be put in list and this list can be updated in the table/map.

```
package hashing;

import java.util.HashMap;
import java.util.Map;

public class MainAnimal {

    /**
     * @param args
     */
    public static void main(String[] args) {

        Animal a1 = new Animal(1);
        Animal a2 = new Animal(2);

        Map<Animal, String> animalsMap = new HashMap<Animal, String>();
```

```
        animalsMap.put(a1,"1");
        animalsMap.put(a2,"2");

        System.out.println(animalsMap.get(a1));

        Map<String, Integer> map = new HashMap<String, Integer>();
        map.put("a", 1);
        map.put("a", 2);// it overrides 1 and puts 2 there

        System.out.println(map.get("a"));

    }

}


class Animal {

    private int index = 0;

    Animal(int index){
        this.index = index;
    }

    public boolean equals(Object obj){
        if(obj instanceof Animal) {
            Animal animal = (Animal) obj;
            if(animal.getIndex()==this.getIndex())
                return true;
            else
                return false;
        }
        return false;
    }

    public int hashCode() {
        return 0;
    }

    public int getIndex() {
        return index;
    }

    public void setIndex(int index) {
        this.index = index;
    }


}
```

In the above code, am showing two different things.

case 1 - two different instances resolving to same hashkey case 2 - two same instances acting as keys for two different entries.

Animal instances, a1 & a2 resolves to same key. But they are not overriden. Hashing mechanism probes through the hash slots and places the entries on different slots.

with the second case, keys resolve to same hash key and also the equals method satisfies. Hence overriding happens.

Now if in the animal class I override the equals method this way -

```
    public boolean equals(Object obj){
//      if(obj instanceof Animal) {
//          Animal animal = (Animal) obj;
//          if(animal.getIndex()==this.getIndex())
//              return true;
//          else
//              return false;
//      }
//      return false;
        return true;
    }
```

Overriding happens. The behavior is like using same instance. Since a1 and a2 are in the same bucket and equals return true as well.

edited Oct 18 '12 at 2:15

answered Oct 18 '12 at 2:05

Larsen

317 2 9