What happens if we do not override hashcode() and equals() in hashmap?

June 2, 2020 (http://javainfinite.com/java/what-happens-if-we-do-not-override-hashcode-and-equals-in-hashmap/) Sri Vikram Sundar (http://javainfinite.com/author/srisundar22/) Java (http://javainfinite.com/category/java/)

What happens if we do not override hashcode() and equals() in hashmap?

In this post, let us understand what is the use of equals() and hashcode(), why it is important to override them and what will happen if we dont override them

To understand this post, it is expected to have knowledge of Internal Working of HashMap.

If not aware of internal working of hashmap, lets have a quick overview.

Important Points of HashMap:

- HashMap has **Key and Value pairs.**
- HashMap is **not Synchronized (i.e. Not Thread Safe)**
- HashMap has **no guarantees as to the order of the map**
- HashMap **does not guarantee that the order will remain constant over time.**
- HashMap allows only **one Null Key**, Which will always be stored at 0th position in the bucket
- HashMap can have **multiple null values**, but **only one null key** is allowed
- HashMap can be **Synchronized externally using Collections.synchronizedMap(HashMap)**

Internal Working of HashMap: (Quick Summary)
So How does HashMap works Internally?
HashMap has the implementation of Map Interface. HashMap has key and value pairs.

The **initial capacity of HashMap is 16** and where as **load factor is 0.75.**

What is Initial Capacity and LoadFactor?
The initial capacity is the number of buckets in the hashtable and the load factor is a measure of how full the hash table is allowed to get before its capacity is automatically increased.

When we try to put a Key and Value within HashMap, first the equals method will check whether both objects are equal and then it hashes and find the bucket where to insert this Key and Value. If both the objects are equal, it will override the already existing Object in Bucket.
In Short, equals() will help us to identify if the object is unique and HashCode helps us to identify the bucket in which the values has to be Stored.

Now Let us see this with an Example,

For the below Example, we are going to create Employee class and another Main class.

```
class Employee {

	String name;

	public Employee(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```

We have created an Employee class, Now let us create a main method

```
public class HashMapExercise {

	public static void main(String args[]) {
		Employee emp1 = new Employee("One");
		Employee emp2 = new Employee("One");

		HashMap<Employee, String> hm = new HashMap<Employee, String>();

		hm.put(emp1, "One");
		hm.put(emp2, "Two");

		System.out.println("Both Objects are Equal: "+emp1.equals(emp2));
		System.out.println("Employee 1 Hashcode: "+emp1.hashCode());
		System.out.println("Employee 2 Hashcode: "+emp2.hashCode());
		hm.forEach((k, v) -> System.out.println("Key is: " + k + " Value is: " + v));
	}

}
```

equals() and hashcode() – Not Overridden:

The above code, we haven't implemented equals() and hashcode() method. Let us the what output our code gives,

```
Both Objects are Equal: false
Employee 1 Hashcode: 366712642
Employee 2 Hashcode: 1829164700
Key is: Employee@15db9742 Value is: One
Key is: Employee@6d06d69c Value is: Two
```

(http://javainfinite.com/wp-content/uploads/2020/05/op1-5.jpg)

Though both objects are equal() since we haven't overridden the equals(), so it considers both the objects as unique keys. We also haven't overridden the hashcode(), so even when both objects are same, their hashcode is different.

How it works without Overriding equals() and hashcode()?

From the output, first it compares two objects (eventhough they are equals, since we have not overridden the equals()) it shows both the objects are not equals and will be considered as Unqiue Keys and with values.

From the hashing, it goes to different buckets – duplicate entries for same object.

Override equals() and Not hashcode:

Now let us Override only Equals() method,

```
@Override
	public boolean equals(Object obj) {
		if (this == obj) {
			return true;
		}
		if (!(obj instanceof Employee)) {
			return false;
		}
		Employee other = (Employee) obj;
		return Objects.equals(name, other.name);
	}
```

Now after overriding equals method, let use slightly modify our main method,

```
Both Objects are Equal: true
Employee 1 Hashcode: 366712642
Employee 2 Hashcode: 1829164700
Key is: Employee@15db9742 Value is: One
Key is: Employee@6d06d69c Value is: Two
```

(http://javainfinite.com/wp-content/uploads/2020/05/op2-1.jpg)

Now after overriding only equals() method, we can see both objects are compared and identified as equals objects (Both Objects are Equal: true)

But here the problem is, though both the objects are equal() the hashcode() is different and so the both objects will be stored in different buckets.

If both objects are equal and if it points to same bucket then the value will be overridden.

Override hashcode() and Not equals():

```
	@Override
	public int hashCode() {
		return Objects.hash(name);
	}
```

When we run the code,

```
Both Objects are Equal: false
Employee 1 Hashcode: 79461
Employee 2 Hashcode: 79461
Key is: Employee@13665 Value is: One
Key is: Employee@13665 Value is: Two
```

(http://javainfinite.com/wp-content/uploads/2020/05/op3-2.jpg)

Since we have overridden only hashcode() and not equals method() –> The objects comparison becomes false, means the objects are unique. So even though the hashcode points to same bucket, the objects are considered as unique keys and both the values will be stored in the bucket.

override both equals() and hashcode():

```
	@Override
	public int hashCode() {
		return Objects.hash(name);
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj) {
			return true;
		}
		if (!(obj instanceof Employee)) {
			return false;
		}
		Employee other = (Employee) obj;
		return Objects.equals(name, other.name);
	}
```

Now let us run the above code and see,

```
Both Objects are Equal: true
Employee 1 Hashcode: 79461
Employee 2 Hashcode: 79461
Key is: Employee@13665 Value is: Two
```

(http://javainfinite.com/wp-content/uploads/2020/05/op4-1.jpg)

Since we have overridden the equals() and hashcode(), when inserting in map — It has identified **both objects are equal** and both has **same hashcode values**. So when trying to insert into the bucket, only **one value will be inserted** that is the reason we have only one element in hashmap (Key is: Employee@13665 Value is: Two)

Complete Code:

```
import java.util.HashMap;
import java.util.Objects;

class Employee {

	String name;

	public Employee(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	@Override
	public int hashCode() {
		return Objects.hash(name);
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj) {
			return true;
		}
		if (!(obj instanceof Employee)) {
			return false;
		}
		Employee other = (Employee) obj;
		return Objects.equals(name, other.name);
	}

}


public class HashMapExercise {

	public static void main(String args[]) {
		Employee emp1 = new Employee("One");
		Employee emp2 = new Employee("One");

		HashMap<Employee, String> hm = new HashMap<Employee, String>();

		hm.put(emp1, "One");
		hm.put(emp2, "Two");

		System.out.println("Both Objects are Equal: "+emp1.equals(emp2));
		System.out.println("Employee 1 Hashcode: "+emp1.hashCode());
		System.out.println("Employee 2 Hashcode: "+emp2.hashCode());
		hm.forEach((k, v) -> System.out.println("Key is: " + k + " Value is: " + v));
	}

}
```


equals and hashcode in java (http://javainfinite.com/tag/equals-and-hashcode-in-java/) equals() and hashcode() (http://javainfinite.com/tag/equals-and-hashcode/)

What happens if we do not override hashcode() and equals() in hashmap? (http://javainfinite.com/tag/what-happens-if-we-do-not-override-hashcode-and-equals-in-hashmap/)

what happens if we dont override equals and hashcode (http://javainfinite.com/tag/what-happens-if-we-dont-override-equals-and-hashcode/)

why we need to override equals and hashcode (http://javainfinite.com/tag/why-we-need-to-override-equals-and-hashcode/)

Why we need to override equals and hashcode in hashmap (http://javainfinite.com/tag/why-we-need-to-override-equals-and-hashcode-in-hashmap/)

One thought to “What happens if we do not override hashcode() and equals() in hashmap?”

ANURAG KUMAR
June 8, 2021 at 10:08 am (http://javainfinite.com/java/what-happens-if-we-do-not-override-hashcode-and-equals-in-hashmap/#comment-5413)

All cases at a place Good demonstration 🙂

REPLY

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

How does hashmap work Internally | Internal Working of HashMap (http://javainfinite.com/java/internal-working-of-hashmap/)

Integrate Shell with SpringBoot – SpringBoot Shell (http://javainfinite.com/spring-boot/integrate-shell-with-springboot-springboot-shell/)

SEARCH

Search...

FOLLOW ME @

Facebook (https://www.facebook.com/JavaInfinite/)

Twitter (https://twitter.com/JavaInfinite)

YouTube (https://www.youtube.com/channel/UCDr7Zrh73kpqTuYoa32DgGA)

(https://www.facebook.com/JavaInfinite/)
(https://twitter.com/JavaInfinite)

(https://www.youtube.com/channel/UCDr7Zrh73kpqTuYoa32DgGA)

RECENT POSTS

Spring Retryable – Retry to consume Restful Services (http://javainfinite.com/spring-boot/spring-retryable-retry-to-consume-restful-services/)

Java 8 Streams – Streams in Java With Examples (http://javainfinite.com/java/java-8-streams-streams-in-java/)

Integrate Shell with SpringBoot – SpringBoot Shell (http://javainfinite.com/spring-boot/integrate-shell-with-springboot-springboot-shell/)

What happens if we do not override hashcode() and equals() in hashmap? (http://javainfinite.com/java/what-happens-if-we-do-not-override-hashcode-and-equals-in-hashmap/)

How does hashmap work Internally | Internal Working of HashMap (http://javainfinite.com/java/internal-working-of-hashmap/)