Sobre hashCode y colecciones en Java

DAM1 Programación 24-25

En el lenguaje de programación Java, un **HashCode** es un identificador de 32 bits que se almacena en un Hash en la instancia de la clase. Toda clase debe proveer de un método `hashCode()` que permite recuperar el Hash Code asignado por defecto por la clase Object.

Los algoritmos de hash eficientes están detrás de algunas de las colecciones más populares en Java, como `HashMap` y `HashSet`. El método `hashCode()` juega un papel importante en estas colecciones y es importante implementarlo correctamente.

1. HashCode() (Java) - Wikipedia, la enciclopedia libre
2. Guide to hashCode() in Java | Baeldung
3. W3Schools. Java String hashCode() Method.
4. Equals() and Hashcode() in Java - Javatpoint
5. Java equals() and hashCode() Contracts | Baeldung
6. ¿Equals y HashCode? Java - Stack Overflow en español
7. How to Implement Java's hashCode Correctly — SitePoint
8. Good hashCode() Implementation - Stack Overflow
9. How to implement hashCode and equals method - Stack Overflow
10. How to implement a compareTo() method when consistent with Equal and hashcode - Stack Overflow
11. How to override equals(), hashcode() and compareTo() for a HashSet - Stack Overflow

hasCode() y colecciones

Un `HashSet` es una colección que no permite elementos duplicados y utiliza el código hash de los elementos para determinar su ubicación en la colección.

Cuando se agrega un elemento a un `HashSet`, se llama al método `hashCode()` del elemento para calcular su código hash. Luego, el `HashSet` utiliza este código hash para determinar la ubicación del elemento en la colección.

Las colecciones basadas en hash como `HashSet` y `HashMap` pueden ser muy eficientes para ciertas operaciones. Por ejemplo, la búsqueda de un elemento en un `HashSet` o la recuperación de un valor asociado a una clave en un `HashMap` tienen un tiempo de ejecución constante en promedio.

Esto se debe a que estas colecciones utilizan el código hash de los elementos para determinar su ubicación en la colección. Cuando se busca un elemento o se recupera un valor asociado a una clave, se calcula el código hash del elemento o clave y se utiliza para acceder directamente a su ubicación en la colección.

Sin embargo, es importante tener en cuenta que el rendimiento de estas colecciones depende en gran medida de la calidad de la implementación del método `hashCode()` de los elementos. Si el método `hashCode()` no está implementado correctamente y produce muchos códigos hash iguales para elementos diferentes, el rendimiento puede disminuir significativamente.

Es importante que el método `hashCode()` esté implementado correctamente para garantizar un comportamiento eficiente y correcto en colecciones basadas en hash como `HashSet`.

Implementar hashCode()

La manera correcta de implementar el método `hashCode()` depende de la clase en la que se está implementando. En general, el método `hashCode()` debe cumplir con ciertas reglas para garantizar un comportamiento correcto en colecciones basadas en hash.

Una forma fácil de implementar el método `hashCode()` es utilizar la función `hash` de la clase `Objects` para calcular el código hash a partir de los campos relevantes del objeto. El método `hash` toma como argumentos los valores de los campos relevantes del objeto y utiliza el algoritmo de hash estándar para calcular un código hash combinado. Este código hash se puede utilizar como valor de retorno del método `hashCode()` del objeto.

Por ejemplo, si tenemos una clase `Person` con campos `firstName` y `lastName`, podríamos implementar el método `hashCode()` de la siguiente manera:

```
@Override
public int hashCode() {
    return Objects.hash(firstName, lastName);
}
```

En este caso, estamos utilizando el método `hash` de la clase `Objects` para calcular el código hash del objeto a partir de los valores de sus campos `firstName` y `lastName`.

Otra forma común de crear un número "único" a partir de varios números es multiplicar por un número primo (generalmente 31 en las clases JDK) y acumular la suma.

hashCode() y equals()

El método `hashCode()` tiene una relación importante con el método `equals()`.

Si sobreescribes el método `equals()`, también debes sobreescribir el método `hashCode()` para conservar el contrato entre ambos métodos: dos objetos iguales deben retornar el mismo valor de hash.

El método `equals()` no llama al método `hashCode()` para determinar la igualdad de dos objetos. Sin embargo, es importante que ambos métodos sean consistentes entre sí para garantizar un comportamiento correcto en colecciones que utilizan hash, como `HashMap` y `HashSet`.

Implementar equals()

Un ejemplo de cómo implementar un método `equals()` consistente con el método `hashCode()` anterior:

```
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null || getClass() != obj.getClass()) {
        return false;
    }
    Person other = (Person) obj;
    return Objects.equals(firstName, other.firstName)
            && Objects.equals(lastName, other.lastName);
}
```

En este ejemplo, estamos sobreescribiendo el método `equals()` para comparar objetos de la clase `Person`. Primero comprobamos si el objeto pasado como argumento es el mismo que el objeto actual (`this == obj`). Si es así, retornamos `true`.

Luego comprobamos si el objeto pasado como argumento es `null` o si su clase es diferente a la clase del objeto actual (`obj == null || getClass() != obj.getClass()`). Si alguna de estas condiciones se cumple, retornamos `false`.

Finalmente, hacemos un casting del objeto pasado como argumento a la clase `Person` y comparamos sus campos relevantes utilizando el método `equals` de la clase `Objects`. Si todos los campos son iguales, retornamos `true`, de lo contrario retornamos `false`.

Es importante tener en cuenta que si sobreescribes el método `equals()`, también debes sobreescribir el método `hashCode()` para conservar el contrato entre ambos métodos: dos objetos iguales deben retornar el mismo valor de hash.

hashCode() y compareTo()

El método `hashCode()` no tiene una relación directa con el método `compareTo()`. El método `hashCode()` se utiliza para obtener un código hash que sirve como un identificador del objeto y se utiliza en colecciones basadas en hash como `HashSet` y `HashMap`.

Por otro lado, el método `compareTo()` se utiliza para comparar objetos y determinar su orden en colecciones ordenadas como `TreeSet` y `TreeMap`. Aunque ambos métodos pueden ser utilizados en colecciones, su propósito y funcionamiento son diferentes.