Actividad 5

Colecciones

Elizabeth López Alanís

PROGRAMACIÓN ORIENTADA A OBJETOS

Agosto – Diciembre 2021

La clase **HashSet** y **HashMap** tienen una ventaja la cual es, que los valores que se van insertando en la colección internamente no tendrán un orden específico, esto se debe a que estas clases realizan un ordenamiento interno mediante el **hashcode** del elemento(Objeto), por lo cual no sabremos que elemento traerá.

Esto nos da una ventaja para nosotros que es poder sobrescribir el método **hashCode()** y **equals()** para así, si tenemos un objeto que tienen sus mismas características y mismo **hashcode** no se agregue a la colección, una de las principales ventajas de este método es la rapidez en la inserción de los elemento si tenemos cientos de elementos que agregar, esta colección es la indicada para este trabajo.

El algoritmo de búsqueda de un elemento en la colección es el siguiente:

1. Encuentra el elemento (Objeto) mediante el **hashcode**. Imagínense varias personas en un salón con la misma edad.
2. Ya que encontró el elemento mediante el **hashcode** se realiza la comparación si el objeto es igual mediante el método **equals()**. Imaginen que en ese mismo salón se está buscando a una persona con un nombre y apellido específico.

Lo primero será definir un objeto de tipo **Alumno**.

```
public class Alumno {
    private int age;
    private String name;

    Alumno(String name, int age){
        this.name = name;
        this.age = age;
    }
}
```

Las variables age y name serán las necesarias para que se comparen los objetos y su **hashcode**. Agregar los métodos **get** y **set** para cada atributo.

Lo siguiente que haremos será insertar los objetos **Alumno** en un **HashSet**.

```
Alumno person1 = new Alumno("Juan",18);
Alumno person2 = new Alumno("Miguel",25);
Alumno person3 = new Alumno("Luis",18);
Alumno person4 = new Alumno("Luis",18);

HashSet personas = new HashSet();
personas.add(person1);
personas.add(person2);
personas.add(person3);
personas.add(person4);
```

Si verificamos el tamaño del **HashSet** alumnos veremos que nos devuelve «4», ya que por defecto el **HashSet** no realiza ninguna comparación.

```
System.out.println(personas.size());
```

Ahora agregamos 4 objetos alumno a un **HashMap**.

```
Map m = new HashMap();

m.put(person1, "valor1");
m.put(person2, "valor2");
m.put(person3, "valor1");
m.put(person4, "mi valor");
```

El método **size()** nos devolverá un tamaño de «4» ya que el **HashMap** tampoco realiza ninguna comparación. De igual manera el método **keySet()** devuelve todos los elementos del **HashMap**.

```
System.out.println(m.size());
System.out.println(m.values());
```

Ahora vamos a cambiar el comportamiento de la clase **Alumno** para que al insertarlo en un **HashMap** o en un **HashSet** se valide si el elemento insertado ya existe.

Sobrescribimos el método **equals()** de tal manera que dos objetos **Alumno** serán iguales si coinciden sus nombres.

```
@Override
public boolean equals(Object o) {
  if (o instanceof Alumnos) {
    Alumnos p = (Alumnos)o;
    return this.name.equals(p.name);
  } else {
    return false;
  }
}
```

De igual manera sobrescribimos el método **hashCode()**. Para generar el **hashcode** utilizamos la variable edad y la longitud del **String**, consiguiendo así un entero.

```
@Override
public int hashCode() {
  return age * this.name.length();
}
```

Ahora, una vez sobrescritos los métodos **hashCode()** y **equals()**, si verificamos el tamaño del **HashSet** alumnos veremos que nos devuelve «3» ya que, si hay un elemento igual, el método **add()** devolverá false.

De igual manera el método **size()** sobre el **HashMap.keySet()** los nombres de 3 alumnos ya que el método **put()** habrá devuelto false al insertar objetos iguales.

Una cosa que hay que tener en cuenta es que si se sobrescribe solo uno de sus métodos ya sea **hashCode()** o **equals()** no tendremos el comportamiento deseado. Hay que sobrescribir los dos.

El método hashcode

Este método viene a complementar al método equals y sirve para comparar objetos de una forma más rápida en estructuras **Hash** ya que únicamente nos devuelve un número entero. Cuando Java compara dos objetos en estructuras de tipo hash (**HashMap**, **HashSet**, etc.) primero invoca al método hashcode y luego el equals. Si los métodos hashcode de cada objeto devuelven diferente hash no seguirá comparando y considerará a los objetos distintos. En el caso en el que ambos objetos compartan el mismo hashcode Java invocará al método **equals()** y revisará a detalle si se cumple la igualdad. De esta forma las búsquedas quedan simplificadas en estructuras hash.