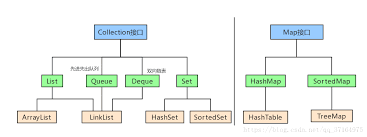
# **Usando las clases HashSet y HashMap**



Antes de iniciar a explicar el codigo, tengo que señalar que tanto la clase [HashSet](http://www.w3api.com/wiki/Java:HashSet" \o "Clase HashSet de Java) como la clase [HashMap](http://www.w3api.com/wiki/Java:HashMap" \o "Clase HashMap de Java) tienen una ventaja la cual es, que los valores que se van insertando en la coleccion internamente no tendran un orden especifico, esto se debe a que **estas clases realizan un ordenamiento interno mediante el hashcode de el elemento(Objeto), por lo cual no sabremos que elemento traerá.**

Esto nos da una ventaja para nosotros que es poder **sobrescribir el metodo**[**.hashCode()**](http://www.w3api.com/wiki/Java:Object.hashCode())**y**[**.equals()**](https://w3api.com/Java/Object-java-lang/equals/)**para asi, si tenemos un objeto que tienen sus mismas caracteristicas y mismo hashcode no se agregue a la colección**, una de las principales ventajas de este metodo es la rapidez en la insercion de los elemento si tenemos cientos de elementosque agregar, esta coleccion es la indicada para este trabajo.

El algoritmo de busqueda de un elemento en la colecion es el siguiente

1. Encuentra el elemento (Objeto) mediante el hashcode. Imagínense varios personas en un salon con la misma edad.
2. Ya que encontro el elemento mediante el hashcode se realiza la comparacion si el objeto es igual mediante el método [.equals()](https://w3api.com/Java/Object-java-lang/equals/). Imaginen que en ese mismo salon se esta buscando a una persona con un nombre y apellido especifico.
3. **Lo primero será definir un objeto de tipo Alumnos.**

public class Alumnos {

private int age;

private String name;

Alumnos(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.**

1. Lo siguiente que haremos será insertar los objetos Alumnos en un [HashSet](http://www.w3api.com/wiki/Java:HashSet" \o "Clase HashSet de Java).

Alumnos person1 = new Alumnos("Juan",18);

Alumnos person2 = new Alumnos("Miguel",25);

Alumnos person3 = new Alumnos**("Luis",18)**;

Son iguales , ojoo!!

No se deberían agregar

Alumnos person4 = new Alumnos**("Luis",18**);

1. **HashSet personas = new HashSet();**

personas.add(person1);

personas.add(person2);

personas.add(person3);

personas.add(person4);

**Si verificamos el tamaño del [HashSet](http://www.w3api.com/wiki/Java:HashSet" \o "Clase HashSet de Java) alumnos veremos que nos devuelve «4», ya que por defecto el [HashSet](http://www.w3api.com/wiki/Java:HashSet" \o "Clase HashSet de Java) no realiza ninguna comparación.**

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

**Ahora agregamos 4 objetos alumno a un [HashMap](http://www.w3api.com/wiki/Java:HashMap" \o "Clase HashMap de Java).**

Map m = new HashMap();

m.put(person1, "valor1");

m.put(person2, "valor2");

m.put(person3, "valor1");

m.put(person4, "mi valor");

El método [.size()](http://www.w3api.com/wiki/Java:HashMap.size()) nos devolvera un tamaño de «4» ya que el [HashMap](http://www.w3api.com/wiki/Java:HashMap" \o "Clase HashMap de Java) tampoco realiza ninguna comparación. De igual manerá el método [.keySet()](http://www.w3api.com/wiki/Java:HashMap.keySet()) devuelve todos los elementos del [HashMap](http://www.w3api.com/wiki/Java:HashMap" \o "Clase HashMap de Java).

System.out.println(m.keySet());

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

Ahora vamos a cambiar el comportamiento de la clase Alumnos para que al insertarlo en un [HashMap](http://www.w3api.com/wiki/Java:HashMap" \o "Clase HashMap de Java) o en un [HashSet](http://www.w3api.com/wiki/Java:HashSet" \o "Clase HashSet de Java) se valide si el elemento insertado ya existe.

**Sobrescribimos el método**[**.equals()**](https://w3api.com/Java/Object-java-lang/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()](http://www.w3api.com/wiki/Java:Object.hashCode()). Para generar el hascode 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()**](http://www.w3api.com/wiki/Java:Object.hashCode())**y**[**.equals()**](https://w3api.com/Java/Object-java-lang/equals/)**, si verificamos el tamaño del [HashSet](http://www.w3api.com/wiki/Java:HashSet" \o "Clase HashSet de Java) alumnos veremos que nos devuelve «3» ya que, si hay un elemento igual, el metodo .[add()](http://www.w3api.com/wiki/Java:HashSet.add()" \o "Método add de la clase HashSet) devolvera false.**

**Map m = new HashMap();**

De igual manera el método [.size()](http://www.w3api.com/wiki/Java:HashMap.size()) sobre el [HashMap](http://www.w3api.com/wiki/Java:HashMap" \o "Clase HashMap de Java).keySet() los nombres de 3 alumnos ya que el método [.put()](http://www.w3api.com/wiki/Java:HashMap.put()) habrá devuelto false al insertar objetos iguales.

System.out.println(m.keySet());

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

Una cosa que hay que tener en cuenta es que si se sobreescribe solo uno de sus metodos ya sea [.hashCode()](http://www.w3api.com/wiki/Java:Object.hashCode()) o [.equals()](https://w3api.com/Java/Object-java-lang/equals/) no tendremos el comportamiento deseado. Hay que sobrescribir los dos.