Colecciones en Java (Collections)

Diferencia entre HashSet y TreeSet

Las dos clases **HashSet** y **TreeSet** implementan la interfaz **Set**. Esto hace que estas dos colecciones sólo almacenen objetos que no se repiten. Si intentas añadir un elemento que ya está incluido, no lo añade, devolviendo **false**. De hecho, esto los hace funcionar como un conjunto.

- HashSet.

El conjunto de datos no se almacena en un orden específico, aunque se garantiza que no hay duplicados.
Realiza todas las operaciones habituales, añadir un elemento, borrarlo, o comprobar si existe un elemento. Si contiene muchos elementos, resulta muy costoso recorrerlos mediante un Iterator.

- TreeSet.

Los elementos del conjunto se almacenan de menor a mayor.
Realiza todas las operaciones anteriores en tiempo logarítmico con el número de elementos

Resumiendo, si queremos tener los objetos ordenados es mejor usar la clase **TreeSet**, pero si el orden no importa debemos usar **HashSet** porque es mas rápido en general.

*Si queremos tener los objetos de la colección ordenados pero vamos a tener que hacer muchas inserciones de objetos podemos implementar una **HashSet** para añadir los objetos y luego convertirla a **TreeSet** para ternerlos ordenados.*

- Métodos comunes a ambas clases.

- *add()* *return boolean*
- *clear()* *return void*
- *clone()* *return Object*
- *contains(Object o)* *return boolean*
- *isEmpty()* *return boolean*
- *iterator()* *return Iterator<E>*
- *remove(Object o)* *return boolean*
- *size()* *return int*
- *spliterator()* *return Spliteratoe<E>*

* Ver API para ver todos los métodos disponibles.

Consideraciones importantes.

Decimos que estas dos clases no admiten elementos repetidos, ahora bien, si los objetos que almacenamos son objetos creados por el programador (es decir, no son objetos primitivos ni String) es necesario sobreescribir los métodos hashCode() y equals(Object o) para que reconozca los objetos iguales de los que no lo son.

Ejemplo.

Si lo que guardamos en la colección son enteros y tenemos un 3, es obvio que si intentamos almacenar otro 3 no lo guarda porque 3=3.

Si lo que guardamos son String y tenemos a "Maria" guardada, e intentamos guardar otra "Maria" pues lo mismo que antes, no la guarda porque "Maria"="Maria"

Pero si lo que guardamos son objetos Usuario con las propiedades
- identificador (int)
- nombre (String)
- email (String)

¿Cómo sabe cuando un Usuario es igual a otro Usuario?.
Para eso tenemos que sobreescribir los métodos citados anteriormente.

Otra consideración importante hace referencia a la propiedad que poseen los objetos TreeSet en cuanto que decimos que ordena los elementos guardados en dicha colección.

La situación es parecida a la explicada anteriormente, pues es fácil comprobar que 13 es mayor que 8 o que "maria" tiene menos letras que "alvaro", pero, entre dos objetos Usuario como el citado anteriormente, ¿Cúal es mayor o menor?
Para ello debemos implementar la interface Comparable del paquete java.lang, de esta manera nos obliga a implementar el único método que tiene que es compareTo(Object o) y de esa menera le decimos cual es el mayor.

Estos dos conceptos los veremos con código para ver como se implementan todos esos métodos.