APRENDERAPROGRAMAR.COM

# INTERFACES SET Y SORTEDSET DEL API DE JAVA. CLASES HASHSET Y TREESET. EJEMPLO. DIFERENCIAS ENTRE ELLAS. (CU00924C)

Sección: Cursos
Categoría: Lenguaje de programación Java nivel avanzado I

Fecha revisión: 2039

**Resumen:** Entrega nº24 curso "Lenguaje de programación Java Nivel Avanzado I".

Autor: Manuel Sierra y José Luis Cuenca

## INTERFACES SET Y SORTEDSET

Vamos a estudiar dos interfaces del paquete java.util del api de java que son muy usadas. Se trata de las interfaces SET y SORTEDSET. Estas interfaces tratan respectivamente lo que se conoce como un conjunto de elementos (SET) y un conjunto de elementos ordenados (SORTEDSET). Debido a su similitud hemos decidido comentarlas conjuntamente en lugar de por separado.


### SET

La interface SET como hemos comentado es la encargada del tratamiento de conjuntos en el api de Java. Un conjunto en Java es una colección de elementos que, como el conjunto en matemáticas, no permite elementos duplicados dentro de ella y no tiene orden entre sus elementos. Más formalmente, no permite elementos e1, e2 tales que: e1.equals(e2) sea true. Además nos obliga a implementar determinados métodos, por ejemplo el método hashCode() entre otros.

### SORTEDSET

Esta interfaz es muy similar a la interface SET. Tan solo se diferencia en que SORTEDSET permite que los elementos dentro del conjunto de la colección estén ordenados totalmente, facilitando por tanto su acceso en búsquedas y haciendo más rápido su consulta.

Los elementos son ordenados usando su orden natural, o bien usando un Comparator. El concepto de orden natural y Comparator los hemos estudiado en apartados anteriores del curso.

### HASHSET

HashSet es la clase que vamos a utilizar para implementar la interfaz SET ya que es quizás la más usada para implementar esta interface. Esta clase implementa la interface SET basada en una tabla hash (a modo resumen para nosotros una tabla hash será una tabla que se construye en base a claves que permiten localizar objetos). Por ejemplo un DNI podría ser la clave para localizar a una persona. En esta clase la clave da la posición del objeto en la tabla, permitiendo un acceso directo al elemento. Este acceso directo hace que esta clase sea ideal para búsqueda, inserción y borrado de elementos en base a una clave o llave. No hay garantía de orden (por ejemplo si hacemos un recorrido de los objetos dentro de un HashSet no siempre los obtendremos en igual orden) y se permite el uso de elementos nulos.

**TREESET**

TreeSet es la clase que vamos a utilizar como implementación de la interface SORTEDSET. Esta implementación está basada en el uso de una estructura de árbol permitiendo que los elementos estén ordenados bien por orden natural o bien por orden total definido por un Comparator. Esto hace muy rápidas las búsquedas, inserciones y borrados de sus elementos. A efectos prácticos, la diferencia principal de esta clase con HashSet es que sus elementos están ordenados. Otra diferencia es la estructura de datos que sirve para almacenar datos, en un caso una tabla y en otro un árbol.

**EJEMPLO DE USO DE SET Y SORTEDSET**

A continuación vamos a crear una clase Persona a partir de la cual vamos a definir un conjunto (set) de personas y otro conjunto ordenado (sortedset) también de personas en nuestro programa principal. Añadiremos las mismas personas al conjunto y al conjunto ordenado, imprimiendo al final todas las personas que tienen ambos conjuntos. Comenzaremos escribiendo el siguiente código de la clase Persona en nuestro editor:

```
/* Ejemplo Interface Set y SortedSet, clase HashSet y TreeSet aprenderaprogramar.com */
import java.util.Objects;

public class Persona implements Comparable<Persona>{
   private int idPersona;      private String nombre;      private int altura;

   public Persona(int idPersona, String nombre, int altura) {
      this.idPersona = idPersona;   this.nombre = nombre;    this.altura = altura;}

   @Override
   public String toString() {
      return "Persona-> ID: "+idPersona+" Nombre: "+nombre+" Altura: "+altura+"\n";  }

   @Override
   public int compareTo(Persona o) { return this.altura-o.altura; }

   @Override
   public int hashCode() { return altura + nombre.hashCode() + idPersona; }

   @Override
   public boolean equals(Object obj) {
      if (obj == null) {   return false;  }
      if (getClass() != obj.getClass()) {  return false;   }
      final Persona other = (Persona) obj;
      if (this.idPersona != other.idPersona) {  return false;    }
      if (!Objects.equals(this.nombre, other.nombre)) {
         return false;
      }
      if (this.altura != other.altura) {     return false;   }
      return true;
   }
}
```

En la anterior definición de clase Persona, se puede ver que hemos implementado la interfaz Comparable y que por tanto vamos a trabajar con el orden natural por simplicidad pero si quisiéramos podríamos implementar la interfaz Comparator como se explicó anteriormente. En el método compareTo hemos definido que una persona "es mayor" que otra si su altura es mayor. Este criterio de orden lo usamos simplemente a modo de ejemplo.

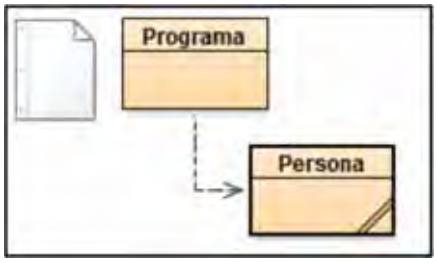
También podemos ver que se han sobreescrito los métodos toString(), hashCode() y equals(), sobreescritura que hemos considerado necesaria para el buen funcionamiento en las interfaces correspondientes. En el método hashCode() hemos implementado que se devuelva un número único como clave. No prestaremos ahora demasiada atención a este método. En el método equals hemos definido un criterio de igualdad entre personas, criterio implementado simplemente a modo de ejemplo.

Escribe ahora el código de nuestro programa que hará uso de las clases HashSet y TreeSet:

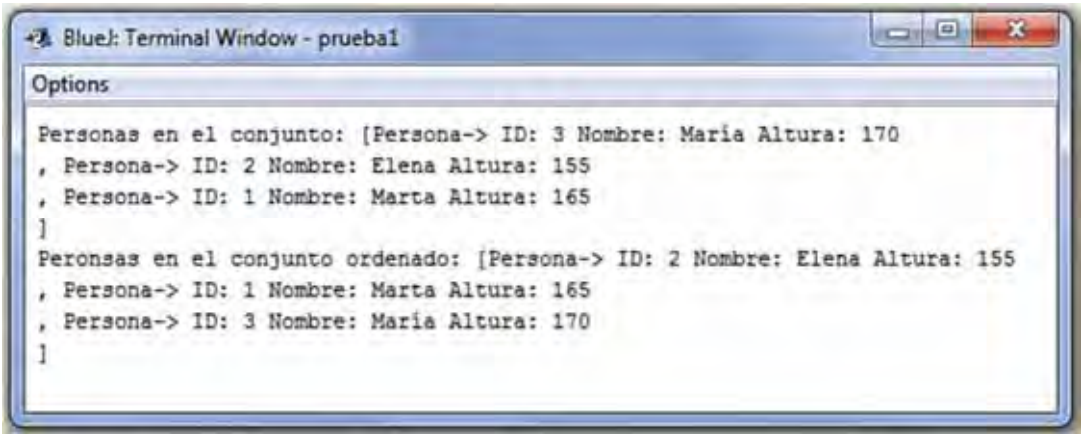
```
/* Ejemplo Interface Set y SortedSet, clase HashSet y TreeSet aprenderaprogramar.com */
import java.util.HashSet;
import java.util.Set;
import java.util.SortedSet;
import java.util.TreeSet;

public class Programa  {
   public static void main (String []args)     {
      Set<Persona> cjsp = new HashSet<Persona>();
      SortedSet<Persona> cjssp = new TreeSet<Persona>();
      Persona p = new Persona(1,"Marta",165);
      cjsp.add(p);
      cjssp.add(p);
      p = new Persona(2,"Elena",155);
      cjsp.add(p);
      cjssp.add(p);
      p = new Persona(3,"María",170);
      cjsp.add(p);
      cjssp.add(p);
      System.out.println("Personas en el conjunto: "+cjsp);
      System.out.println("Personas en el conjunto ordenado: "+cjssp);
   }
}
```

En la clase de nuestro programa podemos ver cómo hemos creado nuestros conjuntos (un set y un sortedset) de personas y hemos añadido a 3 personas (Marta, Elena y María). El diagrama de clases es el siguiente:



El resultado de ejecución del programa nos devuelve la siguiente salida:




Tras observar la salida de nuestro programa vemos que el primer cojunto mantiene los datos desordenados como era de prever, mientras que el conjunto ordenado nos ha introducido a todas las personas ordenadas por su orden natural. En este caso el orden natural viene dado por la altura de las personas, porque así lo hemos definido al implementar el método compareTo. Si hubiéramos definido el orden natural en base a otro criterio (por ejemplo orden alfabético de nombres), la clase SortedSet hubiera devuelto a las personas ordenadas por este otro criterio. Con el criterio empleado la primera persona en el conjunto ordenado es Elena con una altura de 155 centímetros, después Marta con 165 centímetros y por último María con 170 centímetros.

**CONCLUSIONES**

Como conclusión de estas 2 interfaces y estas 2 clases que las implementan podemos decir que la interfaz Set es más sencilla de implementar ya que nos evitamos el tener que usar ordenación, necesario para la interfaz SortedSet. Sin embargo, tenemos un conjunto de elementos de los que no tenemos garantía de orden de ningún tipo y según el caso tendremos que valorar si esto nos resulta conveniente o no. En cambio con SortedSet tenemos el orden establecido bien por orden natural de los elementos contenidos o por un orden total que asignemos con un Comparator, sin que esto suponga demasiado esfuerzo. La sobreescritura de ciertos métodos como hashCode() para implementar HashSet y compareTo() para implementar TreeSet es habitual para conseguir que estas clases funcionen como nosotros queremos. Por el mismo motivo es habitual sobreescribir toString(), ya que si no lo sobreescribimos el resultado obtenido con el método no es adecuado.

**EJERCICIO**

Crea una clase denominada Hotel con los atributos idHotel (int), zona(String), y precio (int). Los valores para zona podrán ser "playa","montaña" o "rural". El precio supondremos que es un dato en euros que podrá tomar valores entre 40 y 150.

Crea una clase con el método main donde se cree un conjunto sin ordenar de 12 hoteles. El programa nos mostrará por consola este conjunto de hoteles y nos preguntará en qué zona queremos el hotel. Tras ésto el programa creará un conjunto ordenado por precio con los hoteles cuya zona corresponda con la zona elegida y los mostrará por consola.

**Ejemplo de ejecución:**

Hoteles en el conjunto:
[Hotel-> ID: 14 Zona: Rural Precio: 146 , Hotel-> ID: 5 Zona: Rural Precio: 92 , Hotel-> ID: 7 Zona: Montaña Precio: 63 , Hotel-> ID: 1 Zona: Playa Precio: 77 , Hotel-> ID: 3 Zona: Playa Precio: 109, Hotel-> ID: 10 Zona: Montaña Precio: 79, Hotel-> ID: 4 Zona: Rural Precio: 43, Hotel-> ID: 6 Zona: Montaña Precio: 142, Hotel-> ID: 11 Zona: Playa Precio: 53, Hotel-> ID: 2 Zona: Montaña Precio: 108, Hotel-> ID: 0 Zona: Rural Precio: 135, Hotel-> ID: 8 Zona: Rural Precio: 66, Hotel-> ID: 9 Zona: Rural Precio: 65]
Elige zona de hotel. (1) Playa (2) Montaña (3) Rural
Elección: 1
Hoteles en el conjunto para la selección "Playa" ordenados por precio:
[Hotel-> ID: 11 Zona: Playa Precio: 53, Hotel-> ID: 1 Zona: Playa Precio: 77, Hotel-> ID: 3 Zona: Playa Precio: 109]

Para comprobar si tu solución es correcta puedes consultar en los foros aprenderaprogramar.com.

**Próxima entrega: CU00925C**

**Acceso al curso completo en aprenderaprogramar.com** -- > Cursos, o en la dirección siguiente: http://aprenderaprogramar.com/index.php?option=com_content&view=category&id=58&Itemid=180