9.C. Conjuntos.

5. Conjuntos (V).

Por defecto, los `TreeSet` ordenan sus elementos de forma ascendente, pero, ¿se podría cambiar el orden de ordenación? Los `TreeSet` tienen un conjunto de operaciones adicionales, además de las que incluye por el hecho de ser un conjunto, que permite entre otras cosas, cambiar la forma de ordenar los elementos. Esto es especialmente útil cuando el tipo de objeto que se almacena no es un simple número, sino algo más complejo (un artículo por ejemplo). `TreeSet` es capaz de ordenar tipos básicos (números, cadenas y fechas) pero otro tipo de objetos no puede ordenarlos con tanta facilidad.

Para indicar a un `TreeSet` cómo tiene que ordenar los elementos, debemos decirle cuándo un elemento va antes o después que otro, y cuándo son iguales. Para ello, utilizamos la interfaz genérica `java.util.Comparator`, usada en general en algoritmos de ordenación, como veremos más adelante. Se trata de crear una clase que implemente dicha interfaz, así de fácil. Dicha interfaz requiere de un único método que debe calcular si un objeto pasado por parámetro es mayor, menor o igual que otro del mismo tipo. Veamos un ejemplo general de cómo implementar un comparador para una hipotética clase "`Objeto`":

```
class ComparadorDeObjetos implements Comparator<Objeto> {

    public int compare(Objeto o1, Objeto o2) { ... }

}
```

La interfaz `Comparator` obliga a implementar un único método, es el método compare, el cual tiene dos parámetros: los dos elementos a comparar. Las reglas son sencillas, a la hora de personalizar dicho método:

- Si el primer objeto (`o1`) es menor que el segundo (`o2`), debe retornar un número entero negativo.
- Si el primer objeto (`o1`) es mayor que el segundo (`o2`), debe retornar un número entero positivo.
- Si ambos son iguales, debe retornar 0.

A veces, cuando el orden que deben tener los elementos es diferente al orden real (por ejemplo cuando ordenamos los números en orden inverso), la definición de antes puede ser un poco liosa, así que es recomendable en tales casos pensar de la siguiente forma:

- Si el primer objeto (`o1`) debe ir antes que el segundo objeto (`o2`), retornar entero negativo.
- Si el primer objeto (`o1`) debe ir después que el segundo objeto (`o2`), retornar entero positivo.
- Si ambos son iguales, debe retornar 0.

Una vez creado el comparador simplemente tenemos que pasarlo como parámetro en el momento de la creación al `TreeSet`, y los datos internamente mantendrán dicha ordenación:

```
TreeSet<Objeto> ts=new TreeSet<Objeto>(new ComparadorDeObjetos());
```

Ejercicio resuelto

¿Fácil no? Pongámoslo en práctica. Imagínate que Objeto es una clase como la siguiente:

```
class Objeto {

    public int a;

    public int b;

}
```

Imagina que ahora, al añadirlos en un `TreeSet`, estos se tienen que ordenar de forma que la suma de sus atributos (`a` y `b`) sea descendente, ¿cómo sería el comparador?

Solución:

Una de las posibles soluciones a este problema podría ser la siguiente:

```
class ComparadorDeObjetos implements Comparador<Objeto> {
```

```
    @Override
    public int compare(Objeto o1, Objeto o2) {
     int sumao1=o1.a+o1.b;  int sumao2=o2.a+o2.b;
     if (sumao1<sumao2) return 1;
     else if (sumao1>sumao2) return -1;
     else return 0;
    }
}
```