**Colecciones {**

-Listas

-Conjuntos

-Mapas

CONJUNTOS <3

-No admite elementos repetidos. el método add comprueba que no haya datos repetidos, si hay algún dato repetido no lo inserta, y devuelve un booleano si lo ha insertado o no.

-Se recorren con un bucle for-each y con un **iterator**

-NO tienen posiciones

Podemos definir la colección con la interfaz pero instanciar la colección con una clase que implementa esa interfaz.



HASHSET

No hay un orden establecido predeterminado.

- Al usar el HASHSET poner el **HashCode & Equals** en la clase necesaria para poder comparar atributos por ejemplo un DNI que es único.

LINKEDHASHSET

Orden de inserción lo hacemos con LINKEDHASHSET.

-El orden está establecido por el orden de inserción de datos,al usar el LINKEDHASHSET poner el **HashCode & Equals** en la clase necesaria.

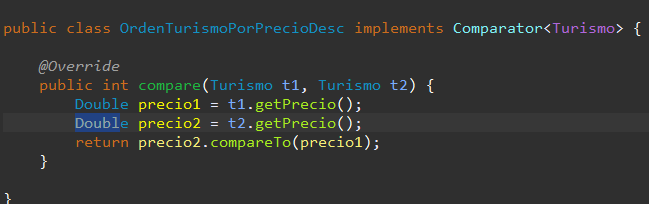
TREESET

El orden será con el criterio especificado, ejemplo: Si es un conjunto de String Alfabéticamente..

-Para ordenar un conjunto podemos usar tree set estableciendo un criterio, pero tambien una lista. Podremos usar tree set para ordenar **SOLO** cuando tengamos el criterio de orden sea un atributo que no se repita, osea que sea único como un DNI.

Si el orden será por un objeto, nosotros tendremos que especificar el tipo ordenación.

Al usar TREESET no podemos ordenar por ejemplo por precio ya que puede hacer precios repetidos, pero si podemos ordenar por la clave primaria porque sabemos que no se repiten.**Como el orden lo hacemos en otra clase usaremos el Comparator<>**

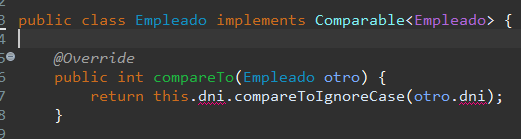


RECORDATORIO: al usar el comparator/comparable usamos **Primitivos**  int - Integer double - Double

Y asi se veria en la clase donde llamo a esta clase(algo que ya hemos hecho antes):



Cuando usamos el TREESET Y queremos ordenar un conjunto por ….. pondremos en la clase en la que ordenaremos un **implements Comparable<>**



MAPAS

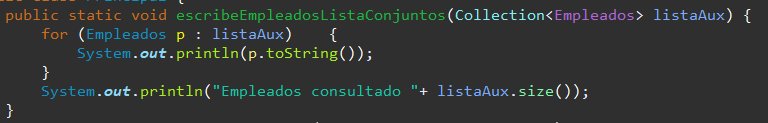
j

—-------------------------

**}**

Collections

Al usar listas y conjuntos como ambos descienden de la clase collections podemos usar este para leer tanto listas como conjuntos

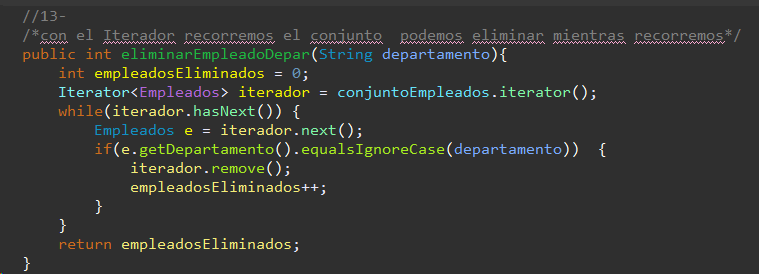


Bucle for- each

NO se puede eliminar un objeto con remove dentro del bucle for-each ya que da error.

Iterator

Al recorrer con un iterator si que podemos usar un remove del iterator



**FIcheros {**

**Tener en cuenta Windows nos oculta las extensiones**

**Windows admite tanto \(propio) como /**

**-Ruta Absoluta** : Representa la ruta completa del recurso, parte del directorio raíz hasta llegar al archivo concreto que se está buscando. *Ejemplo C:\Eclipse*

**-Ruta Relativa**: Desde donde estás, representa solo una parte de la ruta, ya que en ella se tiene en cuenta el directorio actual desde el que se está trabajando. No empiezan por una unidad *Ejemplo C:\* y comienza por una carpeta o directamente el nombre del fichero

**}**

**Escribir en FIcheros{**

**BufferedWriter**

Usamos los flujos para poder acceder a los ficheros, siempre cerramos los flujos a los ficheros.

Al crear un objeto de **BufferedWriter**  que tendrá como parámetros un objeto de la clase **FileWriter** en el cual tendremos como parámetros el fichero que queremos escribir y podemos poner otro parámetro que será un **booleano**, este booleano sirve para decir si escribimos al principio o al final del fichero. Si ponemos **true** escribirá a partir de la ultima línea dejando todo intacto, pero si no escribimos nada por defecto tendrá **false** escribirá por encima de todo el fichero borrando lo que hubiera anteriormente solo la primera vez, luego el reto de lineas lo escribirá seguido

**}**

**Leer FIcheros{**

**BufferedReader**

Usamos los flujos para poder acceder a los ficheros, siempre cerramos los flujos a los ficheros.

Al crear un objeto de **BufferedReader**  que tendrá como parámetros un objeto de la clase **FileReadeer**

**(incompleto)**

**}**

**Flujos {**

**NO puede haber más de 1 flujo para un fichero a la vez.**

**}**