Anexo Ordenacion de objetos- Interface Comparator

Ya hemos visto que la clase `Arrays` tiene un método para ordenar datos, el metodo `Array.sort(parámetros)` donde en parámetros enviamos el vector a ordenar e incluso los limites de los índices a ordenar.

El problema surge cuando nos planteamos cómo ordenamos un vector de objetos.

Esta pregunta no se resuelve con un simple `sort` puesto que un objeto tiene múltiples variables (atributos) por los que ordenar, por tanto habrá que especificar por qué atributo queremos ordenar. No es lo mismo ordenar nombres que edades, alturas o salarios ya que son tipos de datos distintos. Para resolver este problema hay que especificar en el método `sort` un nuevo parámetro que llamaremos `Comparator` y que es una clase que implementa dicha interface donde indicamos el criterio para ordenar. Al ordenar por ese criterio los demás atributos se quedarán ligados al criterio y se reubicaran todos a la vez, puesto que si no fuera así perderíamos la relación entre los datos, es decir, el campo ordenación variaría pero el resto e datos deben acompañarle al moverse en el hecho de la ordenación.

El ejemplo que viene a continuación ilustra muy bien cómo usar `sort` con la interface `Comparator.`

La clase que definimos se llama `Persona` y tiene cuatro atributos `nombre`(String), `edad`(int), `altura`(float) y `DNI`(String de 8 dígitos), su constructor con los cuatro atributos y sus métodos set todos generados por Eclipse.

CLASE PERSONA

```
public class Persona {
   private final String nombre;
   private final int edad;
   private final float altura;
   private final String dni;

   public Persona(String nombre, int edad,float altura,String dni) {
      this.nombre = nombre;
      this.edad = edad;
      this.altura=altura;
      this.dni=dni;
   }
   public String getNombre() {
      return nombre;
   }
   public int getEdad() {
      return edad;
   }
   public float getAltura() {
      return altura;
   }
   public String getDni() {
      return dni;
   }
}
```

CLASE PRINCIPAL

```
import java.util.Random;
import java.util.Arrays;
public class Principal {

   public static void main(String[] args) {
      Persona personas[] =new Persona[100]; //un array de 100 objetos (Persona) desordenados
      // generamos los objetos y los guardamos en el vector
      generaDatos(personas);
      int t;
      //imprimimos por pantalla el array de personas
      System.out.println("\n******Array sin orden:");
      listar(personas);
      //ordenamos con nuestro comparador el array de personas y lo imprimimos por pantalla
      System.out.println("\n******En orden segun el comparador Edad:");
      Arrays.sort(personas, new OrdenPorEdad());
      listar(personas);
      // ordenamos por otro criterio
      System.out.println("\n******En orden segun el comparador DNI:");
      Arrays.sort(personas, new OrdenPorDni());
      listar(personas);
   }//main

   static public void generaDatos(Persona personas[]){// genera el vector
      Random azar = new Random();
      int t,z;
      String dni;
      String nombre[]={"Luis","Carlos","Jose","Eva","Carmen","Luisa","Manuel","Santos"};
      for (t=0;t<personas.length;t++){
         dni="";
         for (z=1;z<=8;z++){
            dni=dni+(char)(azar.nextInt(10)+48);
         }
         personas[t] =new Persona(nombre[azar.nextInt(nombre.length)],
                                  azar.nextInt(10)+40,azar.nextInt(50)+150,dni);
      }
   }//generaDatos

   static public void listar(Persona personas[]){ // metodo estático para imprimir vector
      int t;
      for (t=0;t<personas.length;t++){
         System.out.println(personas[t].getNombre() + "\t" +personas[t].getEdad()
         + "\t"+personas[t].getAltura()+ "\t" +personas[t].getDni());
      }
   }//listar

}//class
```

Resaltado en verde aparece el comparador que se explica a continuación.

Por **cada criterio de ordenación** que se quiera usar hay que **crear una clase** donde se **implemente** la interfaz **Comparator** con el **criterio de ordenación**, en este caso DNI

```
import java.util.Comparator;

public class OrdenPorDni implements Comparator <Persona> {

   public int compare(Persona p1, Persona p2) {
      return p1.getDni().compareTo(p2.getDni());
   }
}
```

Clase donde implementamos el criterio de ordenación por edad:

```
import java.util.Comparator;

public class OrdenPorEdad implements Comparator <Persona> {

   public int compare(Persona p1, Persona p2) {
      return p1.getEdad()-p2.getEdad();
   }

}
```

Las líneas en negrita y letra más grande indican el criterio de comparación del atributo por el que se compara, que en el primer caso es el DNI que es de tipo `String` , estos métodos deben devolver un positivo si el primer argumento es mayor que el segundo , cero si son iguales y un negativo si el segundo es mayor que el primero, por tanto retornan en el caso del String:

`p1.getDni().compareTo(p2.getDni());` `(el método compareTo devuelve >0 , 0, <0 según la comparación)`

y en el caso de la edad, que es un dato numérico:

`p1.getEdad()-p2.getEdad();`

Como ejercicio crear los `Comparator` para ordenar por nombre y por altura y probarlos.