ANEXO: INTERFAZ COMPARABLE

Todos los lenguajes de programación disponen de herramientas para comparar valores.

En el caso de valores de variables y tipos primitivos la comparación se realiza mediante operadores: ==, !=, <, <=, >, >= .

Para comparar objetos se utilizan métodos. Por ejemplo, la clase "*String*" dispone de los métodos "*equals*" y "*compareTo*" entre otros.

Si queremos que los objetos de una clase que hemos definido se puedan comparar tendremos que establecer algún criterio que indique cómo comparar esos objetos.

Existen varias alternativas, pero una de las más utilizadas es hacer que nuestra clase implemente la interfaz "Comparable" estableciendo en su único método "compareTo" cómo queremos ordenar nuestro objeto con respecto a otro objeto del mismo tipo, y para ello compararemos uno o varios de los atributos de esos dos objetos devolviendo los siguientes valores:

- -1, cuando nuestro objeto es anterior al otro.
- 0, cuando ambos objetos son iguales (de cara a su ordenación)
- +1, cuando nuestro objeto es posterior al otro.

Ejemplo: clase Contacto

La clase "Contacto" dispone de los atributos son nombre, ape1, ape2 y tlf., y queremos que a la hora de ordenarlos se utilice el primer apellido, en caso de que este coincida que se utilice el segundo, y se éste también coincide que se utilice el nombre.

Tendremos que implementar la interfaz "*Comparable*" y establecer esos criterios de ordenación en el método "*compareTo*".

```
public class Contacto implements Comparable<Contacto>{

  protected String nombre;
  protected String ape1;
  protected String ape2;
  protected long tlf;

  public Contacto(String nombre, String ape1, String ape2, long tlf) {
    this.nombre = nombre;
    this.ape1 = ape1;
    this.ape2 = ape2;
    this.tlf = tlf;
  }

  (…Resto de métodos…)

  @Override
  public int compareTo(Contacto contacto) {
    if (!this.ape1.equals(contacto.ape1)) {
      return this.ape1.compareTo(contacto.ape1);
    } else if (!this.ape2.equals(contacto.ape2)) {
      return this.ape2.compareTo(contacto.ape2);
    } else {
      return this.nombre.compareTo(contacto.nombre);
    }
  }
}
```

Como vemos en el método "*compareTo*" vamos a comparar nuestro objeto con otro recibido como parámetro.

En primer lugar, comprobamos si el primer apellido es diferente, en cuyo caso nos vale como resultado la comparación entre ese primer apellido de ambos contactos, pero si son iguales pasamos a comparar el segundo apellido haciendo lo mismo, y si estos también son iguales, entonces comparamos los nombres.

Tal y como está descrito el método "*compareTo*" estamos considerando que si dos contactos comparten los dos apellidos y el nombre, son iguales a efectos de ordenación, pero podríamos seguir añadiendo tantos criterios como deseemos.

Hecho esto ya podríamos comparar objetos de tipo "Contacto":

```
TextView etiTexto = (TextView)findViewById(R.id.etiTexto);

Contacto c1 = new Contacto("Ana", "Pérez", "Martín", 655123123);
Contacto c2 = new Contacto("Abel", "Pérez", "Martín", 654113112);

if (c1.compareTo(c2) < 0) {
  etiTexto.setText("c1 va antes que c2");
} else if (c1.compareTo(c2) > 0) {
  etiTexto.setText("c1 va después que c2");
} else {
  etiTexto.setText("c1 y c2 son iguales");
}
```

También podremos ordenar *arrays* de este tipo de objetos utilizando el método "*sort*" de la clase "*Collections*":

```
ArrayList<Contacto> contactos = new ArrayList<Contacto>();
contactos.add(new Contacto("Ana", "Pérez", "Martín", 655123123));
contactos.add(new Contacto("Abel", "Pérez", "Martín", 654113112));
contactos.add(new Contacto("Carlos", "Sanz", "López", 651111222));
contactos.add(new Contacto("Marta", "López", "Rubio", 656212343));
contactos.add(new Contacto("Sara", "Hernández", "Villa", 658990887));

Collections.sort(contactos);

String texto = "";
for (Contacto c: contactos) {
    texto = texto + c.toString() + "\n";
}
etiTexto.setText(texto);
```