9.G. Algoritmos sobre listas y arrays.

2. Algoritmos (II).

En Java hay dos mecanismos para cambiar la forma en la que los elementos se ordenan. Volvamos al ejemplo tratado ya en alguna ocasión en el curso, relativo a un conjunto de artículos y que quisiéramos ordenar por código del artículo. Imagina que tienes los artículos almacenados en una lista llamada "`articulos`", y que cada artículo se almacena en la siguiente clase (fíjate que el código de artículo es una cadena y no un número):

```
class Articulo {

    public String codArticulo; // Código de artículo

    public String descripcion; // Descripción del artículo.

    public int cantidad; // Cantidad a proveer del artículo.

}
```

La primera forma de ordenar consiste en crear una clase que implemente la interfaz `java.util.Comparator`, y en ende, el método método `compare` definido en dicha interfaz. Esto se explicó en el apartado de conjuntos, al explicar el `TreeSet`, así que no vamos a profundizar en ello. No obstante, el comparador para ese caso podría ser así:

```
class comparadorArticulos implements Comparator<Articulo>

{

    @Override

    public int compare( Articulo o1, Articulo o2) { return o1.codArticulo.compareTo(o2.codArticulo); }

}
```

Una vez creada esta clase, ordenar los elementos es muy sencillo, simplemente se pasa como segundo parámetro del método `sort` una instancia del comparador creado:

```
Collections.sort(articulos, new comparadorArticulos());
```

La segunda forma es quizás más sencilla cuando se trata de objetos cuya ordenación no existe de forma natural, pero requiere modificar la clase `Articulo`. Consiste en hacer que los objetos que se meten en la lista o array implementen la interfaz `java.util.Comparable`. **Todos los objetos que implementan la interfaz `Comparable` son "ordenables" y se puede invocar el método `sort` sin indicar un comparador para ordenarlos**. La interfaz `comparable` solo requiere implementar el método `compareTo`:

```
class Articulo implements Comparable<Articulo>{

    public String codArticulo;

    public String descripcion;

    public int cantidad;


    @Override

    public int compareTo(Articulo o) { return codArticulo.compareTo(o.codArticulo); }

}
```

Del ejemplo anterior se pueden denotar dos cosas importantes: que la interfaz `Comparable` es genérica y que para que funcione sin problemas es conveniente indicar el tipo base sobre el que se permite la comparación (en este caso, el objeto `Articulo` debe compararse consigo mismo), y que el método `compareTo` solo admite un parámetro, dado que comparará el objeto con el que se pasa por parámetro.

El funcionamiento del método `compareTo` es el mismo que el método compare de la interfaz `Comparator`: si la clase que se pasa por parámetro es igual al objeto, se tendría que retornar 0; si es menor o anterior, se debería retornar un número menor que cero; si es mayor o posterior, se debería retornar un número mayor que 0.

Ordenar ahora la lista de artículos es sencillo, fíjate que fácil: "`Collections.sort(articulos);`"

Autoevaluación

Si tienes que ordenar los elementos de una lista de tres formas diferentes, ¿cuál de los métodos anteriores es más conveniente?

- ○ Usar comparadores, a través de la interfaz java.util.Comparator.
- ○ Implementar la interfaz comparable en el objeto almacenado en la lista.