Práctica 6a

Ordenaciones usando la interfaz Comparator

En esta práctica introducimos la interfaz `Comparator`, la cual nos permite ordenar una colección de múltiples formas distintas, aparte de la ordenación por defecto que conseguimos en la práctica anterior con la interfaz `Comparable`.

Cuestiones Previas

Copia todo tu código generado en el paquete `P5b` de la práctica anterior, a un nuevo paquete `P6a` y realiza en éste todas las modificaciones necesarias para la práctica.

1 Actividad 1. Ordenando la colección de coches ciudad

En esta actividad ordenamos la colección `cityCars` de dos formas distintas, por matrícula y por nivel de carga de la batería. Para ello:

- **Orden por matrícula**. Usaremos para ello la interfaz `Comparable`:
  - Haz que la clase `Car` implemente el interfaz `Comparable`, añadiéndole el método `compareTo()`, que ordenará dos coches por matrícula en orden ascendente.
  - Crea en la clase `CarDB` el método:

    ```
    public void sortByPlate () {...}
    ```

    y asígnale código para ordenar la colección `cityCars` haciendo uso del anterior interfaz `Comparable` sobre la clase `Car`.

    Nota: recuerda el método `sort()` de la clase `Collections`.

- **Orden por carga de la batería y matrícula**. Usando la interfaz `Comparator` sobre la clase `Car`, ordena la colección `cityCars` por nivel de carga de la batería en orden ascendente (para los coches de combustión, que no tienen batería, y a efectos de ordenación, considera que tienen una batería de valor 0). Si los coches tienen el mismo nivel de batería, ordénalos por matrícula en orden ascendente. Para ello:
  - Crea primero la clase de comparación:

    ```
    public class CarComparatorByBatteryLevelAndPlate ... {...}
    ```

    que implemente la interfaz `Comparator` sobre la clase `Car`, añadiéndole el método `compare()` y asignándole código para implementar el anterior criterio de comparación.

    Nota: lo más sencillo puede ser obtener primero el nivel de batería de cada uno de los dos coches que recibe el método `compare()`, según el tipo de coche (recuerda el operador *instanceof*), y luego aplicar el criterio de comparación teniendo en cuenta los dos niveles de batería.

  - Crea en la clase `CarDB` el método:

    ```
    public void sortByBatteryChargeAndPlate () {...}
    ```

    y asígnale código para ordenar la colección `cityCars` haciendo uso del anterior interfaz.

    Nota: revisa otra vez los diferentes métodos `sort()` de la clase `Collections`.

2 Actividad 2. Guardando las plazas ocupadas del parking por hora de entrada

En esta actividad realizamos las modificaciones necesarias para guardar las líneas del fichero del parking (correspondientes a los coches aparcados) ordenadas por hora de entrada en orden ascendente. Si dos plazas tuviesen la misma hora de entrada deberán ordenarse por coordenada de la plaza en orden ascendente. Para ello:

- De forma similar a la de la actividad 1, crea primero la clase de comparación[1] que implemente la interfaz `Comparator` sobre la clase `CarSpace` y define en ella el método `compare()` asignándole el código adecuado para ordenar por el criterio citado.
- En el método `saveParking`:
  - Declara y crea un `TreeSet` temporal, especificando en el constructor la clase de comparación.
  - Puéblalo con el set `busyCarSpaces` (**nota**: fíjate en los métodos `addAll`).
  - Usa luego el `TreeSet` temporal (que siempre está ordenado, característica principal de un `TreeSet`) como fuente para generar el fichero del parking.

3 Actividad 3. Clase principal

Crea la clase `P6a`, que contiene el método `main` y será la clase principal de la práctica. En el método `main`:

- Recibe y lee seis argumentos, los nombres de seis ficheros:
  - `file1`: un fichero existente con la estructura y contenido de un parking.
  - `file2`: un fichero existente de entradas y salidas.
  - `file3`: el nombre del fichero en el que guardar el resultado de la actualización del parking.
  - `file4`: un fichero existente con los coches de ciudad.
  - `file5`: el nombre del fichero donde guardar la colección de coches `cityCars`.
  - `file6`: el nombre del fichero donde guardar el dibujo del parking.
- Crea un objeto `cdb` de la clase `CarDB` e invoca su método `readCityCarsFile` para leer el fichero `file4` con los coches y que estos se guarden en el array `cityCars` del objeto creado `cdb`.
- Crea un objeto de la clase `Parking` (asígnalo a una variable global `miParking`) a partir del fichero `file1`.
- Actualiza `miParking` a partir del fichero de entradas/salidas `file2` (llamando a `processIO(file2)`).
- Guarda `miParking` invocando el método `saveParking` con `file3`.
- Ordena `cityCars` por batería y matrícula, invocando el método `sortByBatteryChargeAndPlate()` desarrollado en la actividad 1.
- Guarda `cityCars` en el fichero `file5`, invocando el método `saveCarsToFile()`.
- Crea el dibujo del parking invocando el método `toMap()` de la clase `Parking` y salva el resultado en el fichero `file6`.

[1] Trata de usar un nombre para esta clase "al estilo" del usado en el de la actividad 1.