Relación de Ejercicios de Clases y Objetos (Adicional 3): ¡Pokémon!

Con esta relación, practicaremos como funcionan los interfaces y como crear clases comparadoras, mientras creamos una base de datos de Pokémon.

Dada la clase **Pokemon** que os paso en el fichero **pokemon.cs**, crear la clase **Pokedex** contendrá una lista de **Pokemon** y que nos permitirá leerla de un fichero y ordenarla de diferentes formas.

La clase contendrá, para empezar, los siguientes elementos:

- Atributos
  - **listaPokemon**, que será una lista de objetos de la clase **Pokemon**.
- Constructor
  - Al que le pasamos un *string* con el nombre del archivo prn (os paso uno como ejemplo) y lo lee a la lista.
- Métodos
  - **ToString**(), que nos saca un listado de los Pokémon, incluyendo al menos número y nombre.

Una vez hecho esto, deberíamos tener una clase que nos permite leer los datos del fichero y mostrarlos por pantalla.

A continuación: ordenaremos la lista de tres formas diferentes.

1) Método **OrdenaNombre**(): Ordenará la lista por nombre del Pokémon. Esta ordenación la haremos manualmente usando uno de los algoritmos de ordenación de listas o arrays que vimos anteriormente. Para ello, simplemente adaptaremos el algoritmo que ordenaba enteros para que ahora ordene Pokémon por nombre: por ejemplo, en el algoritmo de selección tendremos que escribir una función **min** que nos devuelva el Pokémon más pequeño en orden alfabético mientras que en el caso de la burbuja tendremos que cambiar la comparación de un elemento con el siguiente.

2) Método **OrdenaAtaque**(): Ordenará la lista por ataque de mayor a menor. Para esta ordenación modificaremos la clase **Pokemon** y haremos que herede del interfaz **IComparable<Pokemon>**.

   ```
   class Pokemon : IComparable<Pokemon>
   ```

   Al hacer esto, nos marcará un error. Si le decimos que lo arregle implementando el *interface*, nos creará un método **CompareTo** que tendremos que rellenar. Esta función devolverá **+1** si nuestro Pokémon (**this**) es mayor que el que nos pasan por parámetro (**other**), **0** si son iguales y **-1** si el nuestro es más pequeño.

   Una vez escribamos esta función, simplemente usando el **Sort**() normal de la lista se ordenará por esta forma. Al implementar **IComparable** lo que definimos la ordenación por defecto de los objetos de nuestra clase.

3) Método **OrdenaTipo**(): Ordenará por el tipo del Pokémon (en el mismo orden en el que salen en el *enum*). Si dos Pokémon tienen el mismo tipo, se mirará también el segundo tipo para ordenarlos. Para esta ordenación, crearemos una clase comparadora a la que llamaremos **ComparadorTipo**, que implementará el *interface* **IComparer<Pokemon>**.

```
class ComparadorTipo : IComparer<Pokemon>
```

Similar a **IComparable**, esto nos dará un error que al darle a corregir nos creará el método **CompareTo** correspondiente. Tras rellenar este método, podremos ordenar la lista de Pokémon con el mismo método **Sort**, pero pasándole un objeto de nuestra clase comparadora por parámetro: `l.Sort(new ComparadorTipo())`.

Como podemos ver, la clase **IComparer** es similar a **IComparable** en cuanto a su implementación. La ventaja de **IComparable** es que no hay que crear una clase aparte ni hay que pasarle nada al **Sort**, mientras que la ventaja de **IComparer** es que podemos crear más de una clase comparadora para ordenar de diferentes formas nuestros objetos.

Sin que sirva de precedente, no hace falta crear menú ni nada. Simplemente probar los diferentes métodos que habéis hecho.