Enunciado.

Si hasta el momento hemos utilizado en todos los casos estructuras estáticas (arrays) para almacenar datos, llega el momento de aprovechar las bondades de las estructuras dinámicas y la funcionalidad del API Collections.

Ejercicio 1

En la tarea de la Unidad de Trabajo 7 utilizamos un array para almacenar las cuentas bancarias. Esta estructura provoca que nuestra aplicación esté limitada a utilizar 100 cuentas. Modifica dicho proyecto para:

1. Utilizar una estructura de datos dinámica. Determina, de las trabajadas en los contenidos, cuál sería la más idónea, justificando tu respuesta.

La estructura más idónea sería ArrayList ya que nos permite almacenar una serie de objetos de manera dinámica, que es lo que hacemos actualmente. Además, esto nos permitiría almacenar más de 100 cuentas como antes.

Ejercicio 2

Nos han sugerido una mejora en la aplicación desarrollada en la unidad de trabajo 6, en la que gestionamos un concesionario cuyos vehículos eran insertados en un array. El objetivo es mantener los vehículos ordenados por matrícula en la estructura de datos. El objetivo de este ejercicio es:

- Hacer las modificaciones a la clase `Vehiculo` para que sean objetos comparables por matrícula.
- Modificar la clase `Concesionario` para que utilice una estructura de datos dinámica que mantenga los vehículos ordenados. Determina qué estructura es la más apropiada, justificando tu respuesta (Puedes hacerlo en la misma declaración de la propiedad).

Utilizaremos ArrayList ya que nos permite almacenar tantos Vehículos como recursos tengamos. Además, para ordenarlos utilizaremos el método sort() de java.util.Collections, que ordenará los Vehículos en base a su matrícula, tal y como definimos en el compareTo() de la clase Vehículo.