Ejercicio 1

Una **Lista** es una colección de elementos que se contrae y se expande según la cantidad de elementos (distinto a un Array, que siempre tiene el mismo tamaño), los elementos se pueden ingresar en varias posiciones, produciendo desplazamientos: si se ingresa al comienzo, todos los elementos se corren una posición, si se ingresa a la mitad, los elementos a partir del nuevo elemento, se corren una posición. Si se ingresa al final, no se producen desplazamientos.

Existen dos implementaciones comunes:

- ArrayList
- LinkedList

La primera es conceptualmente más fácil de entender, porque los elementos se ingresan en un array, y de acuerdo a las posiciones que se accedan, los elementos deben ser corridos o no.

La segunda es un poco más compleja, pero más eficiente en temas de memoria. El objeto **Lista** cuenta con un miembro "head", que apunta al primer elemento, siendo este un objeto especial, llamado Nodo que tiene dos componentes: dato y siguiente. El dato contiene la referencia al objeto que se quiere guardar en la **Lista**, el siguiente apunta al siguiente objeto del mismo tipo, que contiene al siguiente elemento. Si siguiente es **null**, entonces es el final de la **Lista**. Si el head de la lista es **null**, la lista está vacía.

A. Realice ambas implementaciones, luego analice cómo se pueden reorganizar la jerarquía de clases para que ambas extiendan de **AbstractList**, y reutilizar funcionalidades.
   Ambas deben contar con la funcionalidad:
   - get devuelve el n-ésimo elemento
   - remove remueve el n-ésimo elemento
   - add agrega un elemento en la posición indicada, o al final
   - size informa la cantidad de elementos
   - isEmpty informa si la lista está vacía
B. Analice en qué casos, conviene utilizar una **LinkedList** o una **ArrayList.**
C. Implemente el patrón Iterador para las colecciones **LinkedList** y **ArrayList.**
D. Agregue a las clases **LinkedList** y **ArrayList** un método **sort** que ordene la lista haciendo uso del **compareTo** de los elementos. ¿Qué debe garantizar de los elementos que se quieran ordenar? ¿Dónde corresponde implementar correctamente este método? No utilizar la clase Collections. Utilice el método **BubbleSort.**
E. Sobrecargue el método **sort** con otro método que reciba un **Comparator** de tal forma que se pueda cambiar el criterio de ordenamiento.
F. Agregue un método **isSorted** que informe si la colección está ordenada o no.
G. Sobrecargue el método **isSorted** para indicar si la colección está ordenada bajo un criterio **Comparator** dado.
H. Generalice las colecciones **LinkedList** y **ArrayList.**

Ejercicio 2

Implemente la clase **Interval**, que representa una secuencia igualmente espaciados de números desde el comienzo (inclusive) y hasta el final(exclusive). Debe contener los siguientes métodos:

- first() : devuelve el primer elemento del intervalo
- last() : devuelve el último elemento del intervalo
- get(int i): devuelve el i-ésimo elemento del intervalo.
- size(): devuelve la cantidad de elementos del intervalo

Implemente el patrón Iterador para los elementos de la colección Interval.
Generalice la clase Interval.

Ejercicio 3

Vuelva a implementar los siguientes ejercicios, aplicando **Collections** donde corresponda:

- Guía 2:
    - Ejercicio 2.
- Guía 3:
    - Ejercicio 4.