Escuela Superior Politécnica del Litoral
Facultad de Ingeniería en Electricidad y Computación

Estructuras de Datos

Laboratorio Iteradores y Comparadores

Introducción

Los iterables y los comparadores juegan un papel crucial en la manipulación y organización de colecciones de datos. Un iterable es una estructura que puede ser recorrida secuencialmente mediante un iterador, permitiendo el acceso controlado y ordenado a sus elementos sin exponer su implementación subyacente. Por otro lado, un comparador es un objeto que permite definir un criterio de comparación personalizado para ordenar o buscar elementos dentro de una colección.

En esta práctica, los estudiantes se enfocarán en comprender el concepto y la importancia de los iterables en las estructuras de datos, así como en aprender a implementar la interfaz **Iterable** de forma manual. Además, explorarán el papel de los comparadores y cómo la interfaz **Comparator** permite definir órdenes naturales para la organización y búsqueda de elementos en una colección. A través de la implementación de la interfaz **Comparator**, los estudiantes aprenderán a ordenar elementos según criterios específicos. Finalmente, aplicarán estos conceptos a problemas prácticos, lo que permitirá afianzar el conocimiento teórico mediante la experiencia práctica.

Las interfaces `Comparator` e `Iterable`

`Iterable`

La interfaz **Iterable** define un único método que debe implementarse:

1. **iterator**: Este método devuelve un objeto de tipo Iterator que permite recorrer la colección.

Implementar esta interfaz permite que una estructura de datos sea recorrida utilizando un iterador, proporcionando una mayor flexibilidad y control sobre el proceso de iteración.

`Comparator`

La interfaz **Comparator** define un único método que debe implementarse:

1. **compare**: Este método toma dos objetos y devuelve un valor entero que indica el orden relativo de los objetos. Devuelve un valor negativo si el primer objeto es menor que el segundo, cero si son iguales, y un valor positivo si el primer objeto es mayor que el segundo.

Algunas observaciones de la implementación

Iterable

Al implementar el método **next** en la clase iteradora interna, es crucial manejar adecuadamente la excepción **NoSuchElementException** para indicar que no hay más elementos disponibles para iterar.

Comparator

La implementación de **compare** debe ser consistente con **equals**. Es decir, **(a.compare(b) == 0)** debería implicar que **a.equals(b)** es verdadero. Esto es especialmente importante al usar colecciones que dependen de la igualdad de los objetos.

Se debe cumplir que:

Reflexividad: **compare(x, x)** debe ser 0.

Simetría: **compare(x, y)** debe tener el signo opuesto de **compare(y, x)**.

Transitividad: Si **compare(x, y)** y c**ompare(y, z)** son ambos mayores que 0, entonces **compare(x, z)** también debe ser mayor que 0.

Actividad

En esta práctica, usted implementará los siguientes métodos de las respectivas clases:

`public int` **`countOccurrences(Iterator<E> it, Comparator<E> comp, E elemento)`**: Cuenta cuántas veces aparece elemento en el iterador it, usando comp para comparar igualdad. Ejemplos: [1, 2, 1, 3] y elemento = 1, devuelve 2

`public boolean` **`isStrictlyIncreasing(Iterator<E> it, Comparator<E> comp)`**: Verifica si los elementos del iterador están en **orden estrictamente creciente** según comp.
Ejemplo: [1, 2, 3, 4] ⟶ true, [1, 1, 2] ⟶ false (porque no es estricto)

`public E` **`findNext(Iterator<E> it, Comparator<E> comp, E elemento):`** Dado un iterador **it**, encuentra el primer elemento después de la primera ocurrencia de **elemento** (usando comp para comparar). Si no se encuentra **elemento** o no hay siguiente, devuelve **null**.
Ejemplo: [1, 2, 3, 4] y buscamos 2, el resultado es 3

`public List<Integer>` **`findMinMax(List<Integer> list, Comparator<Integer> comp)`**: Devuelve un par {min, max} según comp. Ejemplo: [5,3,14,9] -> {3,14}

`public int` **`countUniqueElements(List<Integer> list, Comparator<Integer> comp)`**: Cuenta cuántos elementos únicos hay en la lista usando comp. Ejemplo: [1,2,5,3,1,5] -> 2

`public int` **`countElementsBetween(Iterator<E> it, Comparator<E> comp, E start, E end)`**: Cuenta cuántos elementos del iterador están entre start y end (incluyéndolos) según comp (si no se encuentra start para empezar a contar o no se encuentra end para finalizar de contar devuelve 0). Ejemplo: [1, 2, 3, 4], start = 2, end = 3 ⟶ 2, [1, 2, 3, 4], start = 2, end = 5 ⟶ 0

Consideraciones Finales

Para lograr los métodos solicitados, es posible que deba también implementar algunos de los métodos que están vacíos en el código adjunto. Implemente esos métodos vacíos **únicamente si los utiliza** en sus soluciones de los métodos.

En esta práctica, usted es dueño(a) de su estructura. Por tanto, deberá tomar algunas decisiones respecto al comportamiento de sus métodos. Cuando lo considere oportuno, incluir comentarios en su código para justificar sus decisiones de diseño.