PRÁCTICA 8

Estructura de datos

Grupo 201 GII

María Guzmán Valdezate
Guillermo López de Arechavaleta Zapatero

Contenido

Introducción

En esta práctica, hemos desarrollado varios métodos genéricos para aplicar condiciones de filtrado sobre colecciones, permitiendo seleccionar o eliminar elementos que cumplen ciertos criterios de comparación. Para ello, se ha trabajado tanto con la interfaz Comparable como con Comparator, logrando una mayor flexibilidad para trabajar con distintos tipos de datos.

Además, se ha implementado un método auxiliar, crearSetMismoTipo, que permite mantener el tipo de conjunto original (ya sea un HashSet, TreeSet, o LinkedHashSet) en las colecciones resultantes, lo que contribuye a una mejor integración del resultado con el código cliente.

El objetivo de este informe es analizar la eficiencia algorítmica de estos métodos y reflexionar sobre el impacto de emplear predicados personalizados en el procesamiento de colecciones.

Descripción y Análisis de Métodos

seleccionaPredicado (Collection<E> coleccion, Condicion condicion, E referencia)

- Descripción: Este método filtra los elementos de la colección que cumplen una condición de comparación respecto a un valor de referencia, utilizando la interfaz Comparable<E>. Se itera sobre todos los elementos y se añade al conjunto resultado cada elemento que cumple la condición especificada.
- Complejidad algorítmica: Recorre todos los elementos de la colección y los compara, la operación de compareTo es de O (1), por lo que la complejidad total es de O (n).

eliminaPredicado(Collection<E> coleccion, Condicion condicion, E referencia)

- Descripción: Este método devuelve un conjunto con los elementos de la colección original que no cumplen la condición de comparación con respecto a un valor de referencia. Se usa igualmente la interfaz Comparable<E> para las comparaciones.
- Complejidad algorítmica: Recorre todos los elementos de la colección y los compara, la operación de compareTo es de O (1), por lo que la complejidad total es de O (n).

seleccionaPredicado(Collection<E> coleccion, Condicion condicion, E referencia, Comparator<E> comparador)

- Descripción: Este método devuelve un conjunto con los elementos de la colección original que cumplen la condición de comparación con un valor de referencia, usando un comparador externo (Comparator<E>).
- Complejidad algorítmica: Recorre todos los elementos y aplica una comparación de coste constante, por lo que la complejidad total es O(n).

eliminaPredicado(Collection<E> coleccion, Condicion condicion, E referencia, Comparator<E> comparador)

- Descripción: Este método devuelve un conjunto con los elementos que no cumplen la condición de comparación con un valor de referencia, utilizando un Comparator<E> para evaluar.
- Complejidad algorítmica: Evalúa todos los elementos de la colección con una operación constante, lo que resulta en una complejidad de O(n).

crearSetMismoTipo(Collection<E> coleccion)

- Descripción: Este método devuelve un conjunto vacío del mismo tipo que la colección original, conservando su orden o comparador si lo tuviera.
- Complejidad algorítmica: Solo realiza comprobaciones de tipo y crea el conjunto, por lo que su complejidad es O(1).

Consideraciones adicionales

Dentro de los test hemos tenido que implementar las clases Coche, ComparadorCaballos y GeneradorCoches por que no estaban implementadas aun que así lo ponía en el enunciado. Además, después de revisar varias veces los test hemos comprobado y llegado a la conclusión de que en el método compruebaSeleccionNoComparable hemos cambiado los tamaños resultado porque nos parecía que estaban erróneos, ya que nuestro método sí estaba añadiendo bien los coches.

Conclusiones

La implementación de métodos genéricos con soporte para Comparable y Comparator ha demostrado ser una herramienta poderosa para realizar operaciones de filtrado sobre colecciones de manera flexible y eficiente. Todos los métodos presentan una complejidad lineal, lo que los hace adecuados para trabajar con grandes volúmenes de datos.

Se ha resaltado la utilidad de ofrecer versiones que acepten un Comparator, lo que permite definir distintos criterios de ordenación o comparación sin modificar los objetos a comparar. Esta distinción favorece la reutilización del código y su adaptación a contextos más variados.

Por otro lado, el método auxiliar crearSetMismoTipo garantiza que el conjunto resultante mantenga el mismo tipo que la colección original, asegurando así la coherencia estructural en el tratamiento de datos. Este detalle, aunque sutil, mejora la robustez y la integración del código.

En resumen, la práctica ha permitido comprender cómo aplicar condiciones personalizadas a colecciones de forma genérica, y cómo el diseño cuidadoso de la estructura de datos impacta en la calidad y mantenibilidad del código.