Portfolio Programación
UD 5 - TAD

Jhonal Jesús Roca Hernandez
Programación
1º DAW
2024 - 2025

Ejercicio 4: Comparando implementaciones de Set

Parte 1: Implementación

1. Crea una clase llamada `ComparadorSet` con un método main.
2. Dentro del método main, implementa lo siguiente:

```
import java.util.*;

public class ComparadorSet {
    public static void main(String[] args) {
        // Crear instancias de cada tipo de Set
        Set<String> hashSet = new HashSet<>();
        Set<String> treeSet = new TreeSet<>();
        Set<String> linkedHashSet = new LinkedHashSet<>();

        // Agregar elementos a cada Set
        String[] elementos = {"Java", "Python", "C++",
"JavaScript", "Ruby", "Java"};
        for (String elemento : elementos) {
            hashSet.add(elemento);
            treeSet.add(elemento);
            linkedHashSet.add(elemento);
        }

        // Imprimir cada Set
        System.out.println("HashSet: " + hashSet);
        System.out.println("TreeSet: " + treeSet);
        System.out.println("LinkedHashSet: " + linkedHashSet);
```

```
        // Medir tiempo de inserción para cada Set
        medirTiempoInsercion(new HashSet<>());
        medirTiempoInsercion(new TreeSet<>());
        medirTiempoInsercion(new LinkedHashSet<>());
    }

    private static void medirTiempoInsercion(Set<Integer> set)
{
        long inicio = System.nanoTime();
        for (int i = 0; i < 100000; i++) {
            set.add(i);
        }
        long fin = System.nanoTime();
        System.out.println("Tiempo de inserción para " +
set.getClass().getSimpleName() + ": " + (fin - inicio) + "
ns");
    }
}
```

Parte 2: Análisis

Después de ejecutar el programa `ComparadorSet`, responde a las siguientes preguntas:

1. ¿Qué diferencias observas en el orden de los elementos de cada Set?
    - **LinkedHashSet** : Imprime los valores en el mismo orden que se lo proporcionamos.
    - **TreeSet** : Imprime los valores de manera natural (Orden alfabético).
    - **HashSet** : imprime los valores de cualquier manera ya que no garantiza un orden específico.

2. ¿Por qué crees que el TreeSet ordena los elementos alfabéticamente?
    - organiza los elementos de forma automática usando un árbol, ordenándolos alfabéticamente.

3. ¿Qué Set elimina los elementos duplicados? ¿Por qué ocurre esto?
    - Todos los **Set** eliminan duplicados pero **TreeSet** y **HashSet** lo hacen comparando sus valores, mientras que **LinkedHashSet** también lo hace pero manteniendo el orden de inserción.

4. ¿Cuál es el Set más rápido para insertar elementos? Basándote en los resultados, ¿por qué crees que es así?
    - **HashSet** suele ser el más rápido, ya que usa una tabla hash para almacenar elementos.
    - **TreeSet** es más lento porque mantiene el orden mediante un árbol balanceado.
    - **LinkedHashSet** es un poco menos lento que **HashSet** porque, además de guardar los elementos, también recuerda el orden en que fueron añadidos.

Parte 3: Investigación

Investiga y responde a las siguientes cuestiones:

1. Enumera las principales ventajas y desventajas de cada implementación de Set (HashSet, TreeSet y LinkedHashSet).

Tipo de Set	Ventajas	Desventajas
HashSet	Es el más rápido para inserciones, eliminaciones y búsquedas.	No mantiene el orden de los elementos.
TreeSet	Mantiene los elementos ordenados automáticamente. Permite operaciones de rango (subconjuntos, mayor/menor que).	Es más lento porque usa un árbol para ordenar los datos. No permite null.
LinkedHashSet	Mantiene el orden de inserción y tiene un rendimiento cercano a HashSet.	Usa más memoria debido a la estructura de lista enlazada adicional.

2. Describe al menos dos situaciones prácticas en las que sería más apropiado usar cada tipo de Set. Justifica tu respuesta.

- 1. **HashSet**
- Ejemplo 1: Guardar una lista de nombres sin repetir, sin importar el orden.
- Ejemplo 2: Almacenar códigos únicos de productos en una tienda.
- Justificación: Es muy rápido y no le importa el orden de los elementos.

- 2. **TreeSet**
- Ejemplo 1: Crear una lista de apellidos ordenados alfabéticamente.
- Ejemplo 2: Llevar un ranking de puntajes en un juego, del menor al mayor.
- Justificación: Siempre mantiene los elementos ordenados automáticamente.

- 3. **LinkedHashSet**
- Ejemplo 1: Guardar una lista de canciones favoritas en el orden en que se agregaron.
- Ejemplo 2: Registrar los últimos accesos de usuarios sin duplicados y en el orden en que ingresaron.
- Justificación: Recuerda el orden de inserción y evita elementos repetidos.

Parte 4: Aplicación práctica **(AMPLIACIÓN OPCIONAL - HACER EN CASA)**

Modifica el programa `ComparadorSet` para incluir las siguientes funcionalidades:

1. Crea un método llamado `medirTiempoBusqueda` que:
    - Reciba como parámetro un Set y un elemento a buscar.
    - Mida el tiempo que tarda en encontrar el elemento en el Set.
    - Imprima el tiempo de búsqueda para cada tipo de Set.
2. Implementa un escenario práctico donde se utilice cada tipo de Set de manera apropiada. Por ejemplo:
    - Usa un HashSet para almacenar los códigos únicos de productos en un inventario.
    - Emplea un TreeSet para mantener una lista ordenada de nombres de estudiantes.
    - Utiliza un LinkedHashSet para guardar el historial de páginas visitadas en un navegador web.
3. Para cada escenario:
    - Explica por qué has elegido ese tipo de Set.
    - Demuestra su funcionamiento con un ejemplo de código.
    - Comenta las ventajas que ofrece esa implementación específica para el escenario elegido.

Parte 5: Trabajo con iteradores

Modifica tu programa `ComparadorSet` para incluir estas actividades:

1. Iteración básica
   Crea un método llamado `mostrarElementosConIterador` que:
   - Reciba cualquier tipo de `Set<String>` como parámetro
   - Utilice un `Iterator` para recorrer todos los elementos
   - Imprima cada elemento con el formato: `"Elemento #1: Java"`
   - Prueba el método con los tres tipos de Set y explica:
     - ¿En qué orden se recorren los elementos en cada caso?
       - **HashSet**: Orden aleatorio.
       - **TreeSet**: Orden alfabético.
       - **LinkedHashSet**: Orden de inserción.
     - ¿Coincide este orden con lo visto en las partes anteriores?
       - Sí, **TreeSet** mantiene el orden natural, **LinkedHashSet** conserva el orden de inserción y **HashSet** no garantiza ningún orden.
2. Modificación durante iteración
   Implementa un escenario donde:
   - Crees un `TreeSet` con 10 números enteros aleatorios (1-100)
   - Usando un iterador:
     - Elimina todos los números pares durante la iteración
     - Añade el valor `1000` al conjunto durante la iteración (¿qué ocurre?)
   - Explica:
     - Por qué es necesario usar `iterator.remove()` en lugar de `Set.remove()`
       - **iterator.remove()** es la forma segura de eliminar elementos durante la iteración.
       - **set.remove()** mientras se itera provoca ConcurrentModificationException.
     - Cómo manejarías la excepción `ConcurrentModificationException`
       - Bueno
3. **(AMPLIACIÓN OPCIONAL - HACER EN CASA)** Aplicación en escenarios prácticos
   Modifica los ejemplos de la Parte 4 para:
   - Usar iteradores en al menos dos de tus escenarios
   - Ejemplo con historial de navegación:

```
LinkedHashSet<String> historial = new LinkedHashSet<>();
```

```
// Simular navegación

historial.add("google.com");

historial.add("perplexity.ai");

historial.add("stackoverflow.com");



Iterator<String> it = historial.iterator();

while(it.hasNext()) {

    String pagina = it.next();

    if(pagina.contains("stackoverflow")) {

        System.out.println("Encontrado recurso de ayuda: " +
pagina);

    }

}
```

- Justifica:
  - ¿Por qué es útil el iterador en estos casos?
  - ¿Qué ventajas ofrece respecto a otros métodos de recorrido?

Equals()	
¿Para qué sirve?	Determina si dos objetos son iguales en contenido.
¿De dónde sale?	Es un método de Object, que puede ser sobrescrito en las clases personalizadas.
Relación con hashCode	**Si equals()** devuelve true para dos objetos, sus hashCode() deben ser iguales. **Si equals()** devuelve false, no es obligatorio que los hashCode() sean diferentes, pero es recomendable.
Tipo de parámetro explícito	Un objeto de tipo Object.
Valor de retorno	**true** si los objetos son iguales, **false** en caso contrario.

compareTo()	
¿Para qué sirve?	Compara dos objetos y devuelve un valor que indica su orden relativo.
¿De dónde sale?	Forma parte de la interfaz Comparable<T> en Java.
Relación con equals()	**compareTo** compara orden (<, =, >). **equals** verifica igualdad (true o false).
Tipo de parámetro explícito	Un objeto del mismo tipo (T).
Valor de retorno	**0** si los objetos son iguales. **Negativo** si el objeto actual es menor. **Positivo** si el objeto actual es mayor.