POO Tema 3. Conjuntos de datos

Martín González Dios

3 de noviembre de 2024

1. Conjuntos de datos

Parte importante de la programación es manejar conjuntos de datos sobre los que se realizan operaciones **CRUD** (buscar, insertar, modificar y borrar). La mayoría de lenguajes soportan la representación de conjuntos mediante arrays, pero estas tienen muchas limitaciones.

2. Array

Conjunto de elementos del mismo tipo que ocupan posiciones de memoria consecutivas. Se accede a los elementos mediante un índice que comienza por 0.
Es un objeto para el que **hay que utilizar new, declararlo no es suficiente**, ya que no se reserva memoria (debemos indicar la memoria). Hereda de Object, tiene como atributo público la longitud y es la única estructura que almacena primitivos.
El uso de arrays **debe limitarse a situaciones con tamaños de datos fijos sin eliminaciones** o al utilizar métodos que devuelvan arrays si no es eficiente convertir a otra estructura.

3. Colecciones: Collection

Collection **es una interfaz** que define las operaciones (inserción, borrado y actualización) que debe tener una colección (grupos de datos o elementos). **No es una clase**, por lo que no se puede reservar memoria para ella. **Tampoco se puede recorrer con un índice**, puesto que no está ordenado, **pero si mediante un for-each**, durante el cual no se puede modificar la colección. Tiene funciones como **add, contains, isEmpty**, remove o iterator. Algunas implementaciones no soportan todos los métodos.

4. Colecciones: iteradores

Iterator es una interfaz en la que se definen operaciones para recorrer elementos de una colección. Se utiliza un puntero para iterar, y los **métodos básicos son hasNext, next y remove**. Es una **alternativa al for-each** de igual rendimiento que permite la eliminación de elementos durante la iteración pero que solo se puede utilizar una vez, hasta que el puntero llegue al final.

5. Listas: List

List es una interfaz que soporta operaciones de una colección y define las de lista (get, set o remove). Son colecciones de elementos ordenados según la secuencia en la que fueron introducidos. **Esta orden se mantiene siempre** y se asigna un índice a cada elemento con el que se puede recorrer la lista (for-each también sirve). **Puede tener objetos duplicados**.

6. Listas: Arraylist

Es un **tipo de lista que soporta objetos null y redimensión dinámica automática**, aunque redimensionar con frecuencia empeora el rendimiento (supone incrementar un array mediante una operación de caché de datos). El constructor sin argumentos reserva espacio para 10 elementos, al superar el límite aumenta en otros 10. Se puede recorrer la lista con los índices y modificar el contenido, pero comprobando que los objetos no son null. Las operaciones de acceso tienen timepo lineal O(n). La clase **ArrayList encapsula un array de objetos que se accede con los métodos de List**. **La referencia al array se guarda en la pila, el array y los atributos en el montón**.
El aliasing tiene mayor importancia en los conjuntos de datos. Permite modificar elementos sin cambiar la dirección del conjunto.

7. Conjuntos: Set

Es una **interfaz con las mismas operaciones que Collection**, pero en el que **se pueden repetir los datos** (según el metodo equals, que hay que sobreescribir), **pudiendo haber un único null** (en algunas implementaciones no se permite). Se puede recorrer con un iterador o con un for-each y se deben controlar los objetos que se modifican para que no haya iguales.

8. Conjuntos: Hashset

Implementa la interfaz Set. Es un Wrapper a la clase HashMap (internamente los datos se almacenan como claves de un HashMap). La orden de inserción depende del hash code. El acceso es O(1).

9. Mapas: Maps

Es una interfaz que relaciona una clave que permite acceder a un valor al que está asociado. **Tanto las claves como los valores son objetos (no primitivos)**, y las **claves no pueden estar repetidas** (los objetos si). Algunas implementaciones permiten claves y valores null.
Para localizar y obtener la clave con la que recuperar un valor se usa equals, que debe ser reescrito para la clase de la llave. **La clave debe ser un objeto inmutable**, si no lo es se debe controlar que los atributos utilizados en el equals no se modifiquen. Los métodos más frecuentes son **get, put, containsValue, containsKey o remove**. Algunas implementaciones de Maps no soportan todos sus métodos. Los objetos de Maps no se pueden recorrer directamente, pero se pueden convertir las claves y datos a colecciones o conjuntos y recorrerlos.
La interfaz Map.Entry define métodos para acceder a una entrada del mapa (dupla ¡clave, valor¿).

10. Mapas: HashMap

Es una clase que implementa la interfaz Map en la que los objetos que son **claves están asociados con un código hash** y si es igual se invoca el método equals (mismo procedimiento en HashSet). Esto hace el **acceso más eficiente**, pues muchas comparaciones se hacen entre enteros y no entre objetos. Para generar el hash code se necesita sobreescribir el metodo hashCode (método nativo, depende de cada JVM). El HashMap implementa una tabla hash en la que los datos se ordenan según una función hash. El acceso a los objetos es O(1).
Los métodos equals y hashCode deben usar objetos mutables.

11. Comparativa

Collection	Ordering	Random Access	Key-Value	Duplicate Elements	Null Element
ArrayList	✅	✅	❌	✅	✅
LinkedList	✅	❌	❌	✅	✅
HashSet	❌	❌	❌	❌	✅
TreeSet	✅	❌	❌	❌	❌
HashMap	❌	✅	✅	❌	✅
TreeMap	✅	✅	✅	❌	❌

Figura 1: Características de los datos

Collection	Performance	Default capacity	Empty size	10K entry overhead	Accurately sized?	Expansion algorithm
HashSet	O(1)	16	144	360K	No	x2
HashMap	O(1)	16	128	360K	No	x2
Hashtable	O(1)	11	104	360K	No	x2+1
LinkedList	O(n)	1	48	240K	Yes	+1
ArrayList	O(n)	10	88	40K	No	x1.5

Figura 2: Tamaño de los conjuntos

12. Buenas prácticas

- Almacenar datos sin repetición y sin orden → HashSet
- Acceder a los datos en orden o restricción de memoria sin prioridad de eficiencia → ArrayList
- Almacenar datos sin orden y con identificador unívoco → HashMap