Valentina Colmenares Leguízamo

Actividad evaluativa
Módulo 4: Estructuras de datos

1. **Java.util.Map**

¿Qué es?

Map es una interfaz en el paquete java.util que representa una estructura de datos que mapea claves únicas a valores. Es parte de la colección estándar de Java.

¿Para qué sirve?

Se utiliza para almacenar y manipular pares clave-valor. Cada clave debe ser única, pero los valores asociados pueden ser duplicados.

Funcionalidad principal

Map permite realizar operaciones como agregar, buscar, actualizar y eliminar elementos, garantizando que las claves sean únicas.

Métodos principales

- **put(K key, V value):** Agrega un par clave-valor al mapa. Si la clave ya existe, reemplaza su valor.

```
Map<String, Integer> map = new HashMap<>();
map.put("Alice", 30);
map.put("Bob", 25);
```

- **get(Object key):** Devuelve el valor asociado con la clave proporcionada o null si no existe.

```
int age = map.get("Alice");
```

- **remove(Object key):** Elimina la entrada asociada con la clave dada.

```
map.remove( key: "Bob");
```

- **containsKey(Object key):** Verifica si una clave específica está presente en el mapa.

```
boolean exists = map.containsKey("Alice");
```

- **containsValue(Object value):** Verifica si un valor específico está presente en el mapa.

```
boolean hasValue = map.containsValue(30);
```

- **keySet():** Devuelve un conjunto (Set) de todas las claves.

```
Set<String> keys = map.keySet();
```

- **values():** Devuelve una colección de todos los valores.

```
Collection<Integer> values = map.values();
```

- **entrySet():** Devuelve un conjunto de todas las entradas (pares clave-valor).

```
Set<Map.Entry<String, Integer>> entries = map.entrySet();
```

Java.util.HashMap

¿Qué es?

HashMap es una clase concreta que implementa la interfaz Map. Utiliza una tabla hash para almacenar las entradas clave-valor, lo que permite un acceso eficiente.

¿Para qué sirve?

Es ideal para situaciones donde se necesita acceso rápido a los datos, como buscar, insertar o eliminar elementos con complejidad promedio de O(1).

Funcionalidad principal

Permite almacenar elementos de manera no ordenada y admite null tanto para claves como para valores.

Métodos principales

HashMap hereda todos los métodos de Map. Sin embargo, proporciona una implementación optimizada para ellos. Algunos métodos específicos y características de HashMap incluyen:

- **putIfAbsent(K key, V value):** Inserta un par clave-valor solo si la clave no existe.

```
map.putIfAbsent("Charlie", 40);
```

- **computeIfAbsent(K key, Function<? super K, ? extends V> mappingFunction):** Calcula un valor y lo inserta si la clave no está presente.

```
map.computeIfAbsent( key: "Dan", k -> k.length());
```

- **replace(K key, V oldValue, V newValue):** Reemplaza el valor asociado con una clave solo si coincide con un valor específico.

```
map.replace( key: "Alice", oldValue: 30, newValue: 35);
```

2. GENERICS

Los genéricos permiten que una clase, interfaz o método trabaje con tipos de datos específicos sin que el tipo sea conocido hasta el momento de la ejecución o compilación. Por ejemplo, en lugar de crear varias versiones de una misma clase o método para diferentes tipos de datos (por ejemplo, para Integer, String, Double, etc.), se puede escribir una única implementación que funcione para cualquier tipo.

Funcionalidad

- **Seguridad de tipos:** Los genéricos permiten que los errores relacionados con el tipo de datos se detecten en tiempo de compilación, en lugar de en tiempo

de ejecución. Evitan la necesidad de realizar conversiones explícitas (casting), reduciendo la posibilidad de errores.

- **Reutilización del código:** Es posible crear estructuras de datos o algoritmos que funcionen para múltiples tipos sin duplicar el código.

- **Flexibilidad:** Facilitan la creación de API y bibliotecas genéricas que pueden ser usadas con cualquier tipo de dato.

¿Cómo se utilizan?

- **Genéricos en clases**

  En este ejemplo, T es el parámetro genérico que se sustituye por el tipo concreto (String, Integer, etc.) cuando se crea un objeto de la clase.

```
public class Caja<T> {  4 usages  new *
    private T contenido;  2 usages

    public void guardar(T item) {  2 usages
        contenido = item;
    }

    public T obtener() {  2 usages  new *
        return contenido;
    }
}
```

```
// Uso:
Caja<String> cajaString = new Caja<>();
cajaString.guardar( item: "Hola");
System.out.println(cajaString.obtener());

Caja<Integer> cajaInteger = new Caja<>();
cajaInteger.guardar( item: 123);
System.out.println(cajaInteger.obtener());
```

- **Genéricos en métodos**

  En este ejemplo, <T> indica que el método acepta un tipo genérico como parámetro.

```java
public class Utilidad {  3 usages  new *
    public static <T> void imprimirElemento(T elemento) {
        System.out.println(elemento);
    }
}
```

```java
// Uso:
Utilidad.imprimirElemento("Texto");
Utilidad.imprimirElemento(456);
Utilidad.imprimirElemento(3.14);
```

3. Las estructuras de datos son un componente fundamental en la programación y la computación, ya que permiten organizar, almacenar y gestionar la información de manera eficiente. Existen diversos tipos de estructuras de datos, cada una con características, ventajas y desventajas específicas. Entre las más comunes se encuentran las listas, las pilas y las colas. A continuación, se detalla la definición, el principal método y las diferencias entre estas estructuras.

   Listas

   Una lista es una estructura de datos lineal que permite almacenar una colección de elementos ordenados. Los elementos en una lista pueden repetirse y no necesariamente tienen que ser del mismo tipo de dato. Las listas son extremadamente flexibles, ya que permiten la inserción, eliminación y acceso a los elementos en cualquier posición. Además, estas operaciones suelen tener diferentes niveles de eficiencia dependiendo de la implementación de la lista (por ejemplo, listas enlazadas o arrays).

   Principal método:
   Acceso a elementos: Las listas permiten el acceso directo a un elemento específico mediante su índice. En una lista basada en un array, esta operación tiene una complejidad de O(1), mientras que en una lista enlazada el acceso es secuencial y su complejidad es O(n).

   Diferencia con otras estructuras: Las listas no tienen restricciones sobre el orden en que se agregan o eliminan elementos. Esto las diferencia de las pilas y colas, que imponen restricciones específicas para estas operaciones.

   Pilas

   Una pila es una estructura de datos lineal que sigue el principio LIFO, lo que significa que el último elemento en entrar es el primero en salir. Este comportamiento es útil en aplicaciones donde se necesita un manejo temporal de los datos, como la evaluación de expresiones matemáticas o la gestión de llamadas recursivas en un programa.

Principal método:
push: Agrega un elemento a la cima de la pila.
pop: Elimina y devuelve el elemento en la cima de la pila.
peek: Permite ver el elemento en la cima sin eliminarlo.

Diferencia con otras estructuras: en una pila no se puede acceder directamente a un elemento arbitrario, ya que solo se puede interactuar con el elemento en la cima.

Colas
Una cola es una estructura de datos lineal que sigue el principio FIFO, es decir, el primer elemento en entrar es el primero en salir. Este comportamiento es útil en sistemas que requieren gestión de tareas en orden, como la impresión de documentos o el manejo de procesos en un sistema operativo.

Principal método:
enqueue: Agrega un elemento al final de la cola.
dequeue: Elimina y devuelve el elemento al frente de la cola.
front: Permite ver el elemento al frente sin eliminarlo.

Diferencia con otras estructuras:
Las colas no permiten el acceso directo a elementos intermedios ni operar en el centro de la estructura. Comparadas con las pilas, las colas tienen un enfoque completamente opuesto en cuanto al orden de procesamiento de los elementos.

REFERENCIAS

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Map.html

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/HashMap.html

Lesson: Generics (Updated) (The Java™ Tutorials > Learning the Java Language)