Colecciones en Java

¿Qué son las colecciones?

Tipos

SET

HASHSET

Es una implementación de la interfaz Set que no permite elementos duplicados. Utiliza una función hash para almacenar los elementos, lo que proporciona un acceso rápido y eficiente. No garantiza un orden específico de los elementos.

- Almacena los elementos en una **tabla hash**
- Es la implementación con **mejor rendimiento de todas**
- **No garantiza ningún orden** a la hora de iterar. El orden puede cambiar durante la ejecución del programa.
- **No** permite elementos **duplicados**. Al intentar agregar un elemento duplicado, se reemplaza el existente.
- **Permite valores nulos**

Estructura

```
Set<Tipo> set = new HashSet<>();
```

Ejemplo

```
//declarar el HashSet
Set<String> set = new HashSet<>();

//agregar elementos
set.add("Elemento1");
set.add("Elemento2");

//eliminar elementos
set.remove("Elemento1");

//imprimir HashSet
for (String elemento : set) {
    System.out.println(elemento);
}
```

TREESET

Es una implementación de la interfaz SortedSet que almacena elementos ordenados en orden ascendente. Utiliza una estructura de árbol para lograr un acceso rápido y mantener los elementos ordenados. Los elementos deben ser comparables o se debe proporcionar un comparador personalizado.

- Almacena los elementos **en orden ascendente según el orden natural de los elementos o mediante un comparador personalizado**
- Es más lento que HASHSET
- **No** permite elementos **duplicados**.
- **NO permite valores nulos**
- Los elementos almacenados deben de **implementar la interfaz Comparable**

Estructura

```
Set<Tipo> set = new TreeSet<>();
```

Ejemplo

```
//declarar
Set<String> set = new TreeSet<>();

//agregar elementos
set.add("Elemento1");
set.add("Elemento2");

//eliminar elementos
set.remove("Elemento1");

//imprimir
for (String elemento : set) {
    System.out.println(elemento);
}
```

LINKEDHASHSET

Es una implementación de la interfaz Set que mantiene el orden de inserción de los elementos. Combina un hash table con una lista doblemente enlazada para mantener el orden. Proporciona un acceso rápido y mantiene el orden de inserción, pero no permite elementos duplicados

- Almacena los elementos **orden de inserción de los elementos, lo que significa que el orden en que se agregan los elementos se mantiene constante.**
- Es más lento que HASHSET
- **No** permite elementos **duplicados**.
- **Permite valores nulos**
- Los elementos almacenados deben de **implementar la interfaz Comparable**

Estructura

```
Set<Tipo> set = new LinkedHashSet<>();
```

Ejemplo

```
//declarar el
Set<String> set = new LinkedHashSet<>();

//agregar elementos
set.add("Elemento1");
set.add("Elemento2");

//eliminar elementos
set.remove("Elemento1");

//imprimir
for (String elemento : set) {
    System.out.println(elemento);
}
```

LIST

ARRAYLIST

Es una implementación de la interfaz List que utiliza un arreglo dinámico para almacenar elementos. Permite el acceso rápido a los elementos por índice, pero puede ser costoso en términos de rendimiento al realizar inserciones o eliminaciones en posiciones intermedias.

- Implementado mediante un arreglo dinámico, lo que permite un **acceso aleatorio eficiente** a los elementos.
- **Permite elementos duplicados**
- **Permite valores nulos**

Estructura

```
List<Tipo> list = new ArrayList<>();
```

Ejemplo

```
//declarar el
List<String> list = new ArrayList<>();
```

```
//agregar elementos
list.add("Elemento1");
list.add("Elemento2");

//eliminar elementos
list.remove("Elemento1");

//imprimir
for (String elemento : list) {
    System.out.println(elemento);
}
```

LINKEDLIST

- Implementado mediante una lista doblemente enlazada, lo que permite un **acceso rápido a los elementos en los extremos de la lista**
- **Permite elementos duplicados**
- **Permite valores nulos**

Estructura

```
List<Tipo> list = new LinkedList<>();
```

Ejemplo

```
//declarar el
List<String> list = new LinkedList<>();

//agregar elementos
list.add("Elemento1");
list.add("Elemento2");

//eliminar elementos
list.remove("Elemento1");

//imprimir
for (String elemento : list) {
    System.out.println(elemento);
}
```

MAP

HASHMAP

Es una implementación de la interfaz Map que almacena pares clave-valor. Utiliza una función hash para acceder rápidamente a los valores correspondientes a una clave. No garantiza un orden específico de los

elementos

- Almacena los **pares clave-valor en una tabla hash**, lo que permite un **acceso rápido** a los valores a través de las claves.
- **No garantiza un orden específico** para los pares clave-valor. El orden puede cambiar durante la ejecución del programa.
- **Permite una clave nula y múltiples valores nulos**.

Estructura

```
Map<ClaveTipo, ValorTipo> map = new HashMap<>();
```

Ejemplo

```
//declarar el
Map<String, Integer> map = new HashMap<>();

//agregar elementos
map.put("Clave1", 1);
map.put("Clave2", 2);

//eliminar elementos
map.remove("Clave1");

//imprimir
for (Map.Entry<String, Integer> entry : map.entrySet()) {
    System.out.println("Clave: " + entry.getKey() + ", Valor: " +
entry.getValue());
}
```

TREEMAP

Es una implementación de la interfaz SortedMap que almacena pares clave-valor ordenados por la clave. Utiliza una estructura de árbol para lograr un acceso rápido y mantener los elementos ordenados. Las claves deben ser comparables o se debe proporcionar un comparador personalizado

- Mantiene los pares clave-valor **ordenados según el orden natural de las claves o mediante un comparador personalizado**.
- **No permite claves duplicadas**. Al intentar agregar una clave duplicada, se reemplaza el valor existente.
- **No permite claves nulas**
- **Permite múltiples valores nulos**.

Estructura

```
Map<ClaveTipo, ValorTipo> map = new TreeMap<>();
```

Ejemplo

```
//declarar el
Map<String, Integer> map = new TreeMap<>();

//agregar elementos
map.put("Clave1", 1);
map.put("Clave2", 2);

//eliminar elementos
map.remove("Clave1");

//imprimir
for (Map.Entry<String, Integer> entry : map.entrySet()) {
    System.out.println("Clave: " + entry.getKey() + ", Valor: " +
entry.getValue());
}
```

LINKEDHASHMAP

Es una implementación de la interfaz Map que mantiene el orden de inserción de los elementos. Combina un hash table con una lista doblemente enlazada para mantener el orden. Proporciona un acceso rápido y mantiene el orden de inserción

- Mantiene el **orden de inserción** de los pares clave-valor, lo que significa que el orden en que se agregan se mantiene constante.
- **No permite claves duplicadas**. Al intentar agregar una clave duplicada, se reemplaza el valor existente.
- **Permite una clave nula y múltiples valores nulos**.

Estructura

```
Map<ClaveTipo, ValorTipo> map = new LinkedHashMap<>();
```

Ejemplo

```
//declarar el
Map<String, Integer> map = new LinkedHashMap<>();

//agregar elementos
map.put("Clave1", 1);
map.put("Clave2", 2);
```

```
//eliminar elementos
map.remove("Clave1");

//imprimir
for (Map.Entry<String, Integer> entry : map.entrySet()) {
    System.out.println("Clave: " + entry.getKey() + ", Valor: " +
entry.getValue());
}
```