ACADEMIA JAVA

P1- COLLECTIOS EN JAVA - XIDERAL: SEMANA 2

NOMBRE: JOSE MANUEL JIMENEZ HDZ

¿QUE ES UNA COLECCION EN JAVA?

Una colección representa un grupo de objetos. Esto objetos son conocidos como elementos. Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos. En Java, se emplea la interfaz genérica Collection para este propósito. Gracias a esta interfaz, podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes, como pueden ser: añadir, eliminar, obtener el tamaño de la colección… Partiendo de la interfaz genérica Collection extienden otra serie de interfaces genéricas. Estas subinterfaces aportan distintas funcionalidades sobre la interfaz anterior. Las colecciones son las siguientes SET, MAP, LIST Y QUEDE.

SET

```
Set set = Collections.synchronizedSet(new
HashSet());
```

```
SortedSet sortedSet =
Collections.synchronizedSortedSet(new TreeSet());
```

```
Set set = Collections.synchronizedSet(new
LinkedHashSet());
```

La interfaz Set define una colección que no puede contener elementos duplicados. Esta interfaz contiene, únicamente, los métodos heredados de Collection añadiendo la restricción de que los elementos duplicados están prohibidos. Es importante destacar que, para comprobar si los elementos son elementos duplicados o no lo son, es necesario que dichos elementos tengan implementada, de forma correcta, los métodos equals y hashCode. Para comprobar si dos Set son iguales, se comprobarán si todos los elementos que

los componen son iguales sin importar en el orden que ocupen dichos elementos.

Dentro de la interfaz Set existen varios tipos de implementaciones realizadas dentro de la plataforma Java. Vamos a analizar cada una de ellas:

HashSet: *esta implementación almacena los elementos en una tabla hash. Es la implementación con mejor rendimiento de todas, pero no garantiza ningún orden a la hora de realizar iteraciones. Es la implementación más empleada debido a su rendimiento y a que, generalmente, no nos importa el orden que ocupen los elementos. Esta implementación proporciona tiempos constantes en las operaciones básicas siempre y cuando la función hash disperse de forma correcta los elementos dentro de la tabla hash. Es importante definir el tamaño inicial de la tabla ya que este tamaño marcará el rendimiento de esta implementación.*

TreeSet: *esta implementación almacena los elementos ordenándolos en función de sus valores. Es bastante más lento que HashSet. Los elementos almacenados deben implementar la interfaz Comparable. Esta implementación garantiza, siempre, un rendimiento de log(N) en las operaciones básicas, debido a la estructura de árbol empleada para almacenar los elementos.*

LinkedHashSet*: esta implementación almacena los elementos en función del orden de inserción. Es, simplemente, un poco más costosa que HashSet.*

LIST:

```
List list = Collections.synchronizedList(new
ArrayList());
List list = Collections.synchronizedList(new
LinkedList());
```

La interfaz List define una sucesión de elementos. A diferencia de la interfaz Set, la interfaz List sí admite elementos duplicados. A parte de los métodos

heredados de Collection, añade métodos que permiten mejorar los siguientes puntos:

- *Acceso posicional a elementos: manipula elementos en función de su posición en la lista.*
- *Búsqueda de elementos: busca un elemento concreto de la lista y devuelve su posición.*
- *Iteración sobre elementos: mejora el Iterator por defecto.*
- *Rango de operación: permite realizar ciertas operaciones sobre rangos de elementos dentro de la propia lista.*

Dentro de la interfaz List existen varios tipos de implementaciones realizadas dentro de la plataforma Java. Vamos a analizar cada una de ellas:

ArrayList: *esta es la implementación típica. Se basa en un array redimensionable que aumenta su tamaño según crece la colección de elementos. Es la que mejor rendimiento tiene sobre la mayoría de situaciones.*

LinkedList: *esta implementación permite que mejore el rendimiento en ciertas ocasiones. Esta implementación se basa en una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento.*

MAP:

```
Map map = Collections.synchronizedMap(new
HashMap());

SortedMap mortedMap =
Collections.synchronizedSortedMap(new TreeMap());

Map map = Collections.synchronizedMap(new
LinkedHashMap());
```

La interfaz Map asocia claves a valores. Esta interfaz no puede contener claves duplicadas y; cada una de dichas claves, sólo puede tener asociado un valor como máximo.

Dentro de la interfaz Map existen varios tipos de implementaciones realizadas dentro de la plataforma Java. Vamos a analizar cada una de ellas:

HashMap: *esta implementación almacena las claves en una tabla hash. Es la implementación con mejor rendimiento de todas, pero no garantiza ningún orden a la hora de realizar iteraciones. Esta implementación proporciona tiempos constantes en las operaciones básicas siempre y cuando la función hash disperse de forma correcta los elementos dentro de la tabla hash. Es importante definir el tamaño inicial de la tabla ya que este tamaño marcará el rendimiento de esta implementación.*

TreeMap: *esta implementación almacena las claves ordenándolas en función de sus valores. Es bastante más lento que HashMap. Las claves almacenadas deben implementar la interfaz Comparable. Esta implementación garantiza, siempre, un rendimiento de log(N) en las operaciones básicas, debido a la estructura de árbol empleada para almacenar los elementos.*

LinkedHashMap: *esta implementación almacena las claves en función del orden de inserción. Es, simplemente, un poco más costosa que HashMap.*