RESUMEN TALLER SOBRE COLECCIONES EN JAVA OPENWEBINARS

1.- DEFINICIÓN DE COLECCIONES

Las colecciones en java se definen como interfaces que nos brindan la capacidad de almacenar elementos y, mediante el uso de diferentes métodos, hacer muchas operaciones diferentes con ellos (añadir, borrar o modificar elementos, saber el número de los mismos, etc). En java disponemos de una gran variedad de tipos de colecciones, las cuales han ido siendo añadidas con la mayoría de actualizaciones de este lenguaje. Las más importantes serán explicadas a continuación:

2.- TIPOS E IMPLEMENTACIONES DE COLECCIONES

-Set: Es una interfaz que define una colección que no admite elementos duplicados. Al heredar de collections, tiene los métodos de este añadiendo únicamente la función de descartar los elementos duplicados, para lo cual comprueba todos los elementos sin tener en cuenta el orden que ocupen los elementos.

-HashSet: Esta implementación de Set utiliza una tabla Hash para almacenar los elementos. Es la implementación que más rendimiento tiene entre las derivadas del Set, pero no establece ningún orden en los diferentes elementos de la tabla. Por este motivo es una de las más usadas cuando no es necesario tener en cuenta el orden de los elementos.

-LinkedHasSet: Esta implementación es básicamente un HashSet ampliado para que se tenga en cuenta el orden de inserción de los elementos. Por ello es un poco menor en rendimiento que su predecesora, aunque sigue superando a la implementación TreeSet.

-TreeSet: Esta es la implementación de Set con el menor rendimiento, pues, debido a su estructura de árbol, utiliza log(N) para sus operaciones básicas.

-List: La interfaz List se define como una sucesión de elementos. Una de las principales diferencias con la interfaz Set es que List si permite el uso de elementos duplicados y, por norma general, tanto esta como sus implementaciones son las más habituales de encontrar. Como en Set, hereda de collections, por lo que tiene todas sus funciones así como añadir las suyas propias las cuales son el acceso posicional a los elementos, la búsqueda de elementos concretos, una mejora en Iterator y permite el uso de ciertas operaciones extra.

-Vector: Es una implementación de List que esta presente desde la primera versión de java. Por tanto, está muy desfasado (legacy), no tiene un rendimiento muy bueno y por tanto no es muy recomendable usarlo en la actualidad

-ArrayList: Es la implementación más usada en Java. Esta basada en un array redimensionable que va creciendo (o decreciendo) en función de los elementos que añadamos (o borremos). En la mayoría de situaciones es la que presenta el mejor rendimiento.

-LinkedList: Esta implementación se utiliza para aumentar el rendimiento en casos muy concretos. Se trata de una lista que esta doblemente enlazada entre elementos, teniendo cada uno un puntero al anterior y al siguiente elemento.

-Otras implementaciones de List:

-Queue: Como indica su nombre, se trata de una fila de elementos que van desplazándose a medida que el primer elemento va cambiando, es decir, todos los elementos añadidos se insertan al final de la cola y avanzan cuando el primer elemento se mueve o elimina.

-Deque: Es una queue doble que puede funcionar en ambos sentidos.

-Map: Esta interfaz tiene la peculiaridad de que no hereda de collections, por tanto solo tiene los métodos que la misma incluye. Esta interfaz asocia a cada valor una clave, no puede contener elementos (claves) duplicados y cada clave solo puede tener asociado un elemento. Las implementaciones que incluye son bastante parecidas a las de la interfaz Set.

-HashMap: Esta implementación de Map almacena las claves en una tabla Hash. Al igual que en HashSet, es la implementación con el mejor rendimiento de entre las 3 principales, a costa de no garantizar un orden fijo en la tabla Hash.

-LinkedHashMap: Al igual que el LinkedHashSet, es una ampliación con menor rendimiento que HashMap, pues permite definir un orden a la tabla Hash de las claves.

-TreeMap: De nuevo como el TreeSet, es la implementación de menor rendimiento pues almacena las claves en función de sus valores usando log(N) en sus operaciones más básicas.

También existen colecciones especiales que se utilizan en casos muy específicos:

-Colecciones no modificables: Son colecciones que si se intentan modificar el programa lanzará un error (*UnsupportedOperationException*) y se pueden utilizar por ejemplo para el resultado de una operación (es decir, un resultado fijo).

-Colecciones sincronizadas: Se utilizan para trabajar de forma pareja con otros hilos en curso del programa.

A su vez, también existen diferentes implementaciones menos usadas que pueden ser muy útiles en determinadas situaciones:

-EnumSet: Se trata de un set ideado para contener valores de enumeración.

-CopyOnWriteArraySet/List: Este Set/List nos permite crear una colección thread-safe, pero sin estar sincronizada.

-EnumMap: Es un Map de alto rendimiento cuyas claves son valores de enumeración.

-WeakHashMap: Es un HashMap que permite referencias débiles a sus claves. Por tanto, si una clave no se va a utilizar más, el recolector de basura se encarga de borrar el par

-IdentityHashMap: Es un HashMap cuyo uso no es común que cambia la comparación de claves con *equals* (=) a ==.

3.- ALGORITMOS UTILIZADOS EN COLECCIONES

Son las operaciones que se pueden realizar con las colecciones y vienen incluidas en la clase *Collections* (diferente de *Collection*). Aunque existen muchas, solo veremos las más importantes:

4.- ALGORITMOS MÁS UTILIZADOS

Algoritmos de ordenación:

-*Collections.sort(list):* Esta operación tiene que recibir una colección del tipo *List.* La colección que recibe queda modificada por dicha operación.

-*Collections.sort(list, comparator):* Esta operación ordena la lista que recibe en el orden del *comparator.*

Algoritmos de búsqueda:

-*Collections.binarySearch():* Esta operación devolverá el elemento que este en la posición especificada, siempre y cuando la lista este ordenada en el orden natural (ascendente).

También existen otras operaciones como hallar el máximo, el mínimo, la frecuencia, disjoint, etc.

También existen librerías de terceros (librerías creadas por empresas/personas que no trabajan directamente para Java) que pueden ser muy útiles en según que casos. Las más utilizadas son las de Google (Guava), las de Eclipse y las de Apache.