9.B. Colecciones.

1. Introducción a las colecciones.

¿Qué consideras una colección? Pues seguramente al pensar en el término se te viene a la cabeza una colección de libros o algo parecido, y la idea no va muy desencaminada. **Una colección a nivel de software es un grupo de elementos almacenados de forma conjunta en una misma estructura**. Eso son las colecciones.

Las colecciones definen un conjunto de interfaces, clases genéricas y algoritmos que permiten manejar grupos de objetos, todo ello enfocado a potenciar la reusabilidad del software y facilitar las tareas de programación. Te parecerá increíble el tiempo que se ahorra empleando colecciones y cómo se reduce la complejidad del software usándolas adecuadamente. Las colecciones permiten almacenar y manipular grupos de objetos que, a priori, están relacionados entre sí (aunque no es obligatorio que estén relacionados, lo lógico es que si se almacenan juntos es porque tienen alguna relación entre sí), pudiendo trabajar con cualquier tipo de objeto (de ahí que se empleen los genéricos en las colecciones).

Además las colecciones permiten realizar algunas operaciones útiles sobre los elementos almacenados, tales como búsqueda u ordenación. En algunos casos es necesario que los objetos almacenados cumplan algunas condiciones (que implementen algunas interfaces), para poder ser hacer uso de estos algoritmos.

Las colecciones son en general elementos de programación que están disponibles en muchos lenguajes de programación. En algunos lenguajes de programación su uso es algo más complejo (como es el caso de C++), pero en Java su uso es bastante sencillo, es algo que descubrirás a lo largo de lo que queda de tema.

Las colecciones en Java parten de una serie de interfaces básicas. Cada interfaz define un modelo de colección y las operaciones que se pueden llevar a cabo sobre los datos almacenados, por lo que es necesario conocerlas. La interfaz inicial, a través de la cual se han construido el resto de colecciones, es la interfaz `java.util.Collection`, que define las operaciones comunes a todas las colecciones derivadas. A continuación se muestran las operaciones más importantes definidas por esta interfaz, ten en cuenta que `Collection` es una interfaz genérica donde "`<E>`" es el parámetro de tipo (podría ser cualquier clase):

- Método `int size()`: retorna el número de elementos de la colección.
- Método `boolean isEmpty()`: retornará verdadero si la colección está vacía.
- Método `boolean contains (Object element)`: retornará verdadero si la colección tiene el elemento pasado como parámetro.
- Método `boolean add(E element)`: permitirá añadir elementos a la colección.
- Método `boolean remove (Object element)`: permitirá eliminar elementos de la colección.
- Método `Iterator<E> iterator()`: permitirá crear un iterador para recorrer los elementos de la colección. Esto se ve más adelante, no te preocupes.
- Método `Object[] toArray()`: permite pasar la colección a un array de objetos tipo Object.
- Método `containsAll(Collection<?> c)`: permite comprobar si una colección contiene los elementos existentes en otra colección, si es así, retorna verdadero.
- Método `addAll (Collection<? extends E> c)`: permite añadir todos los elementos de una colección a otra colección, siempre que sean del mismo tipo (o deriven del mismo tipo base).
- Método `boolean removeAll(Collection<?> c)`: si los elementos de la colección pasada como parámetro están en nuestra colección, se

eliminan, el resto se quedan.

- Método `boolean retainAll(Collection<?> c)`: si los elementos de la colección pasada como parámetro están en nuestra colección, se dejan, el resto se eliminan.

- Método `void clear()`: vacíar la colección.

Más adelante veremos cómo se usan estos métodos, será cuando veamos las implementaciones (clases genéricas que implementan alguna de las interfaces derivadas de la interfaz `Collection`).