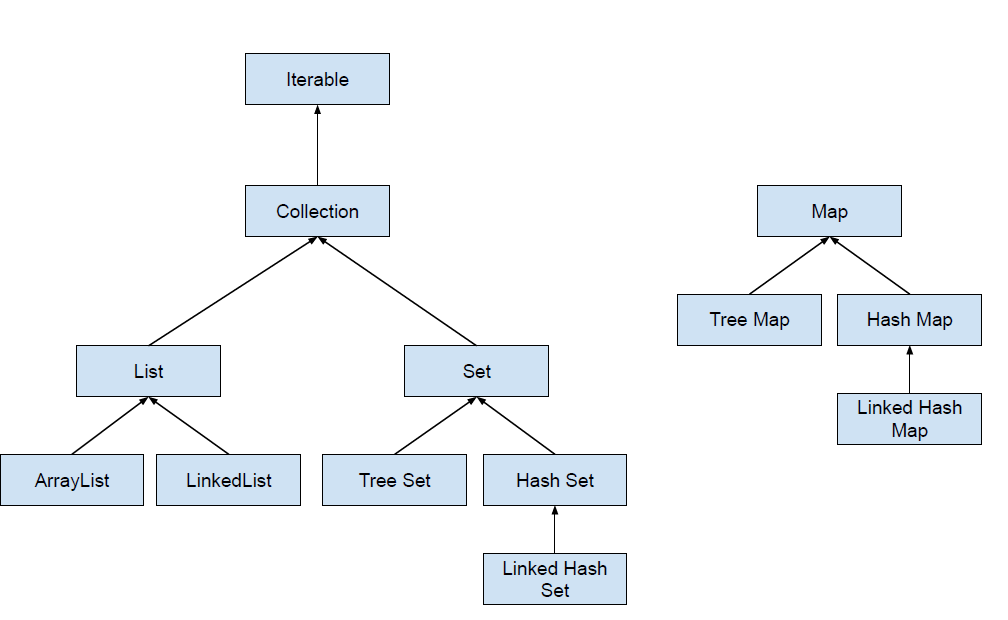
## **Framework de Colecciones de Java**.

****

El Framework de Colecciones de Java es un conjunto de interfaces y clases que proporcionan arquitecturas reutilizables para almacenar y manipular grupos de objetos.

### Iterable y Collection

* **Iterable<T>:** Es la interfaz raíz de toda la jerarquía de colecciones. Su única función es proporcionar un Iterator, lo que permite que cualquier clase que la implemente pueda ser recorrida con un bucle "for-each".
* **Collection<E>**: Es la interfaz principal del framework. Extiende Iterable y representa un grupo de objetos, conocidos como sus elementos. Define los métodos básicos que todas las colecciones deben tener, como add(), remove(), size(), contains() e isEmpty(). No tiene una implementación directa, sino que se especializa en dos sub-interfaces principales: List y Set.

### List: Colecciones Ordenadas con Duplicados

La interfaz **List<E>** representa una **colección ordenada** (también llamada secuencia) que **permite elementos duplicados**. Los elementos se pueden acceder a través de su índice numérico (posición en la lista).

Principales implementaciones:

* **ArrayList<E>**: Implementa la lista utilizando un **array dinámico**. Es muy eficiente para el acceso aleatorio a elementos (mediante get(index)), pero más lento para inserciones o eliminaciones en medio de la lista, ya que puede requerir el desplazamiento de elementos.
* **LinkedList<E>:** Utiliza una **lista doblemente enlazada**. Es muy rápida para añadir o eliminar elementos al principio o al final de la lista. El acceso a un elemento por su índice es más lento que en ArrayList, ya que debe recorrer la lista hasta encontrarlo.

### Set: Colecciones sin Duplicados

La interfaz **Set<E>** representa una colección que **no permite elementos duplicados**. Modela la abstracción matemática de un conjunto.

Principales implementaciones:

* **HashSet<E>:** Utiliza una tabla hash para almacenar los elementos. Es la implementación más rápida, ofreciendo un rendimiento de tiempo constante (O(1)) para las operaciones básicas (add, remove, contains). **No garantiza ningún orden** en los elementos.
* **LinkedHashSet<E>:** Es una subclase de HashSet que, además de la tabla hash, mantiene una lista enlazada. Esto le permite mantener el **orden de inserción** de los elementos. Su rendimiento es casi idéntico al de HashSet.
* **TreeSet<E>:** Almacena los elementos en un árbol (específicamente un Red-Black Tree). Mantiene los elementos en un **orden ascendente** (orden natural de los elementos o según un Comparator especificado). Sus operaciones son más lentas (O(log n)) que en HashSet.

### Map: Estructura de Clave-Valor

La interfaz **Map<K, V>** no extiende de Collection y representa una estructura de datos completamente diferente. Un Map almacena pares **clave-valor**, donde cada clave debe ser única. Es ideal para buscar, actualizar o eliminar un elemento basándose en su clave.

Principales implementaciones:

* **HashMap<K, V>:** Utiliza una tabla hash. Es la implementación más común y ofrece el mejor rendimiento (O(1)) para inserciones y búsquedas. **No garantiza ningún orden** en sus pares clave-valor.
* **LinkedHashMap<K, V>:** Subclase de HashMap que también mantiene una lista enlazada de las entradas. Esto le permite conservar el **orden de inserción**.
* **TreeMap<K, V>:** Utiliza un árbol Red-Black Tree. Mantiene las entradas **ordenadas según la clave**. Es más lento (O(log n)) que HashMap.