# Introducción a las colecciones

## ¿Qué son?

Las colecciones son un **almacén dinámico de objetos**. Al ser dinámico, este almacén puede aumentar o disminuir durante la ejecución del programa. Esta es la principal diferencia con los Arrays, los cuales tenían un tamaño fijo declarado.

Las ventajas de las colecciones frente a los Arrays son:

- Pueden cambiar de tamaño dinámicamente.
- Pueden ir ordenados
- Se puede insertar y eliminar elementos

Sin embargo, una colección solo almacena objetos pero no tipos primitivos (int, float,..)

En la mayoría de los lenguajes de programación existen estructuras para almacenar colecciones de datos. Esto es una serie de datos agrupados a los que se puede hacer referencia con un único nombre.

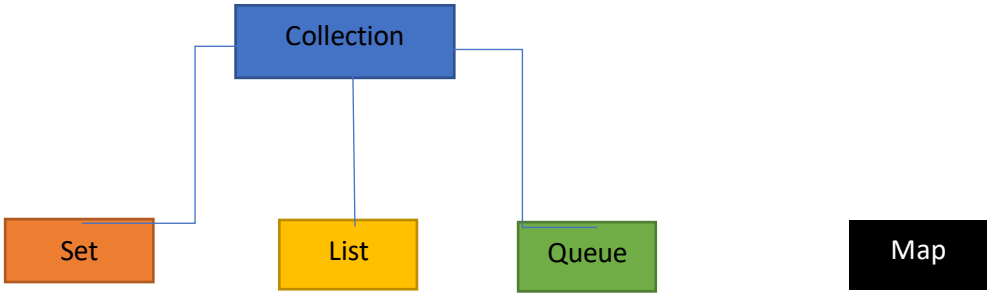
- **Listas** cuando es una enumeración de registros de datos los cuales pueden ser eliminados o aumentados en cualquier momento. Las hay simplemente enlazadas que sólo se pueden recorrer desde el principio al final y sólo en esa dirección; las doblemente enlazadas permiten recorrer la lista en cualquier dirección. Hay también listas circulares en las que no hay un nodo final de la lista porque éste apunta al primero de la lista generando una lista sin fin.
- **Pilas** cuando los datos se almacenan de modo que la última información añadida a la pila es la primera que se obtendrá cuando se recupere información de la pila (se las conoce también como estructuras LIFO, Last In First Out, el último que entra es el primero que sale).
- **Colas** se trata de una estructura en las que los datos se almacenan de modo que al obtener información se consigue obtener el primer dato de la cola (estructura FIFO, First In First Out, el primer que entra es el primero que sales).
- **Árboles**, en este caso los datos se enlazan formando una estructura de en forma de árbol invertido (al estilo de la estructura de los directorios y archivos). Se utiliza mucho para conseguir que los datos estén permanentemente ordenados
- **Tablas Hash**, se utilizan para conseguir índices que permiten asociar claves con valores. Son muy utilizadas en las bases de datos
- **Grafos**. Se llama así a cualquier estructura de datos donde la información se relaciona de forma libre (sin seguir ninguna de las estructuras anteriores).

Por tanto, una colección representa un grupo de objetos. Estos objetos son conocidos como elementos.

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.

Para trabajar con las colecciones debemos conocer el framework que trabaja con las colecciones. Cuando nos referimos a framework nos estamos refiriendo a todas las clases e interfaces, en este caso las correspondientes a las colecciones.

Todas las interfaces surgen de la interfaz Collection de la cual surgen otras, como se puede observar en la siguiente imagen:




- Una colección de la interfaz Set permite almacenar una colección de elementos no repetidos y sin ordenar.
- Una colección de la interfaz List permite almacenar una colección de elementos que pueden estar repetidos y que están indexados por su posición. List permite hacer más cosas que un Set, sin embargo es más lento
- Una colección de la interfaz Queue (colas) permite almacenar una colección de elementos y solamente podemos acceder a los que están al principio, al final o ambos. Similar a la cola de personas en un supermercado.
- Una colección de la interfaz Map (no desciende de Collection) permite almacenar una colección de elementos (puede haber repetidos) indexados por una clave única. Al contrario de los List, en una colección Map la clave no tiene por qué ser numérica.

A continuación, se incluyen los links a la ayuda oficial para revisar cómo se implementan y los métodos disponibles

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

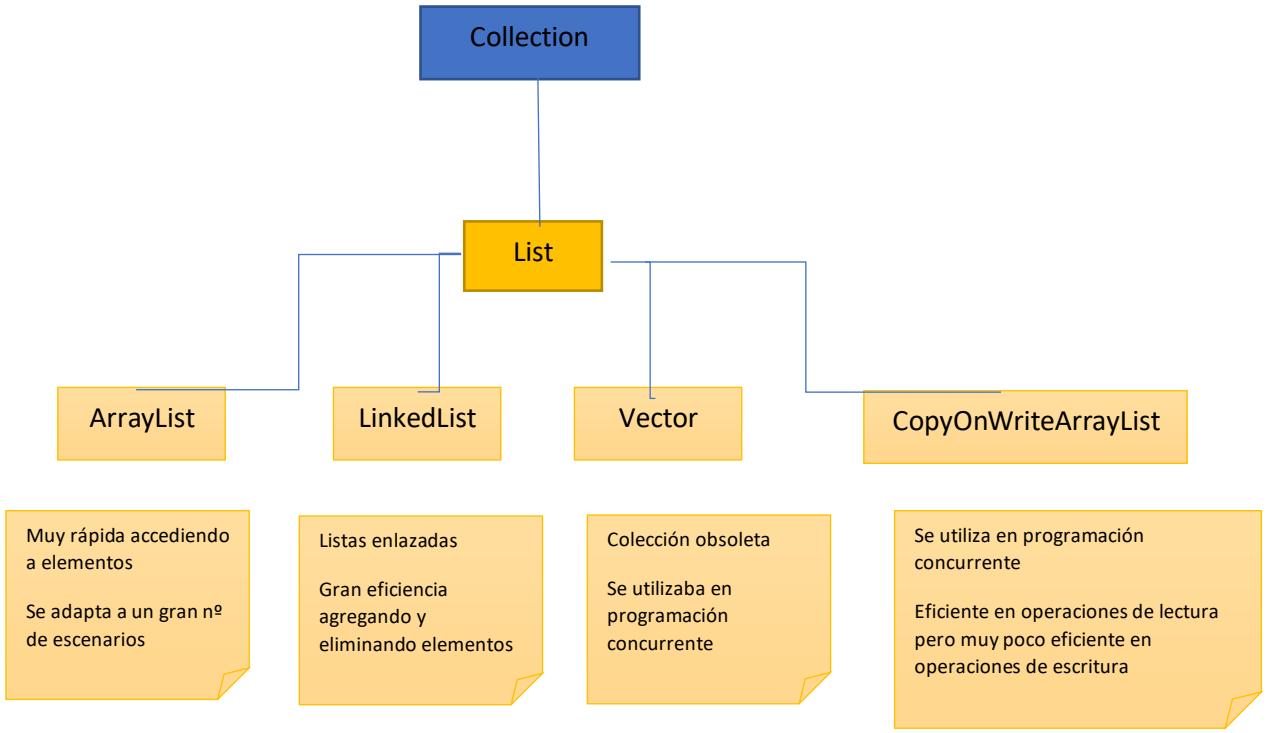
https://docs.oracle.com/javase/8/docs/api/java/util/List.html

https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

A continuación, se expondrán las clases que implementan las interfaces vistas anteriormente. Se comenzará exponiendo algunas clases de la API de Java que implementan la interfaz List.

## Interfaz List




Las ventajas:

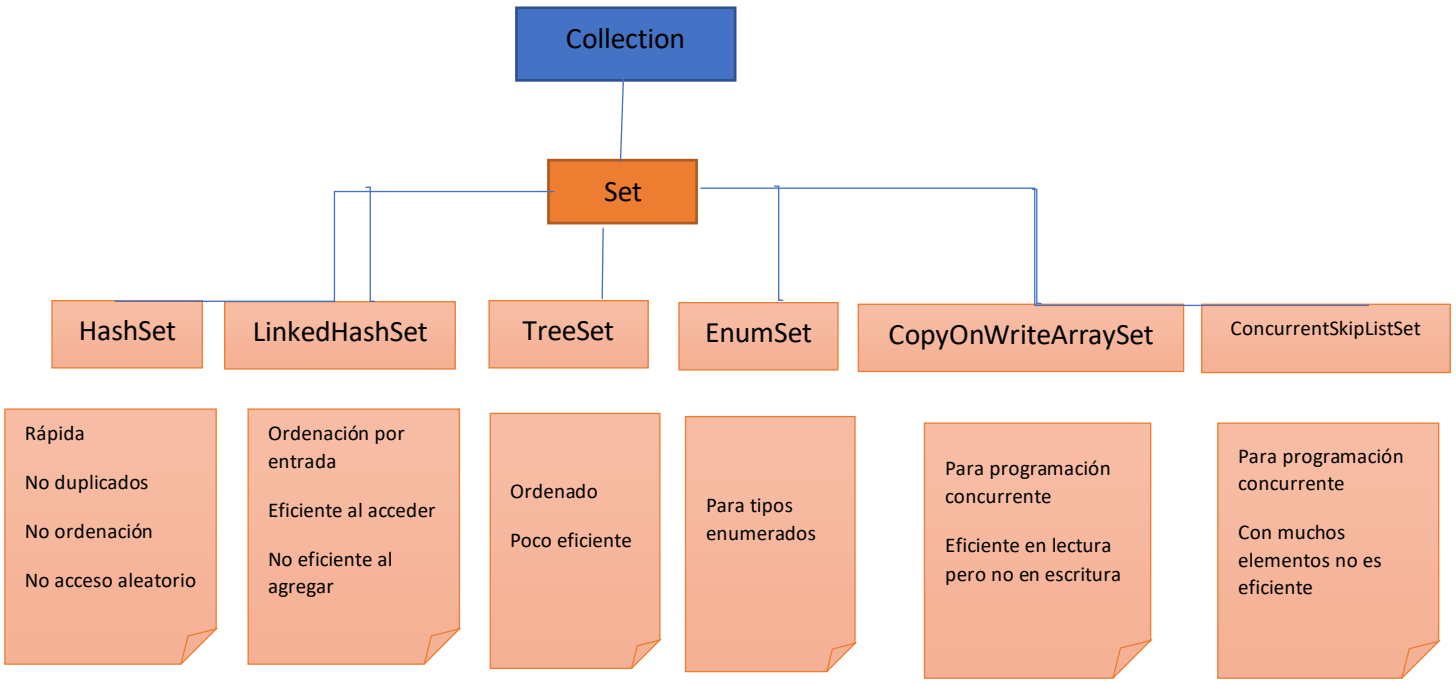
- Acceso a cualquiera de sus elementos
- Están ordenadas (collection.sort())
- Es posible añadir y eliminar elementos
- Tenemos disponible el iterador ListIterator modifica cualquier dirección
- Sintaxis similar a los Arrays

Las desventajas:

- Bajo rendimiento en algunas operaciones que sería mejor usar otras interfaces.

Dentro del uso de las listas, casi siempre se usarán las clases ArrayList o LinkedList, usar las otras clases

## Interfaz set



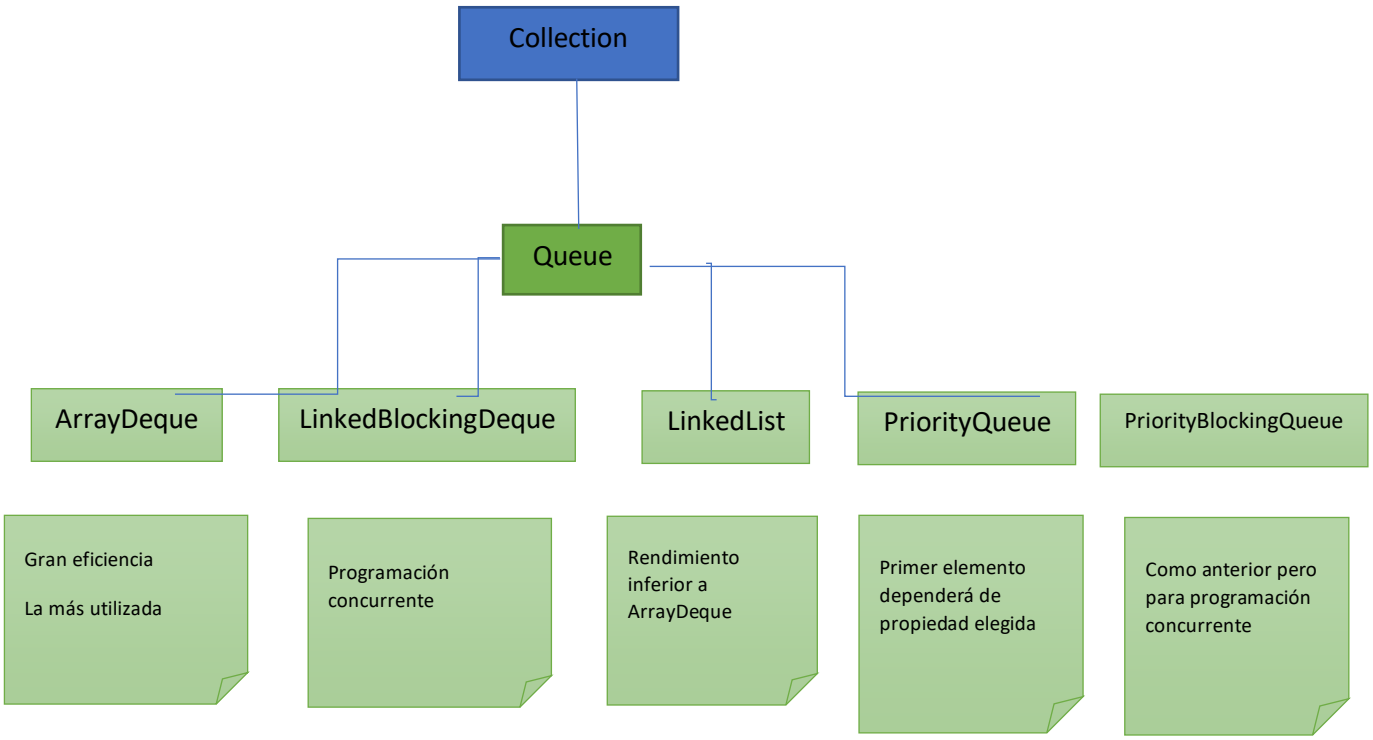

Las ventajas:

- No permiten elementos duplicados
- Uso sencillo del método add que además asegura no elementos duplicados

Las desventajas:

- No tienen acceso aleatorio
- Poca eficiencia a la hora de ordenar elementos

## Interfaz Queue (colas)



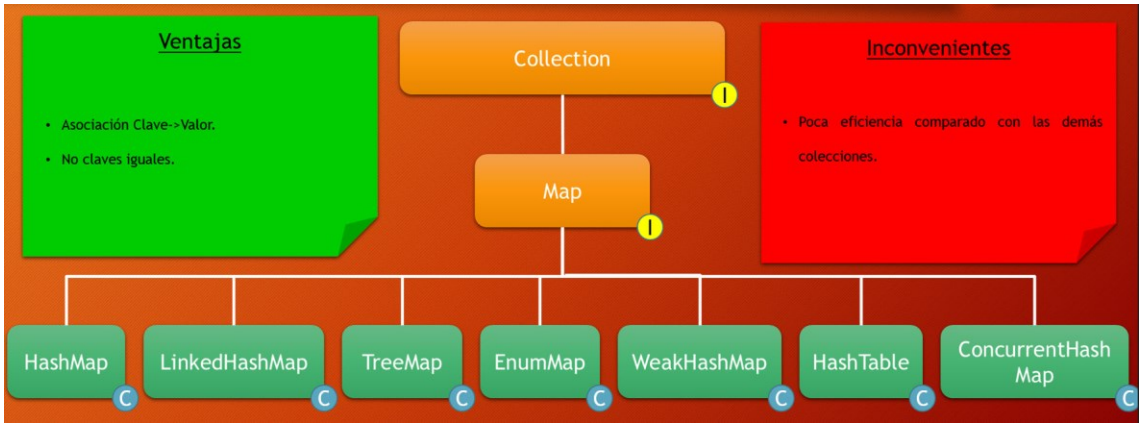

Las ventajas:

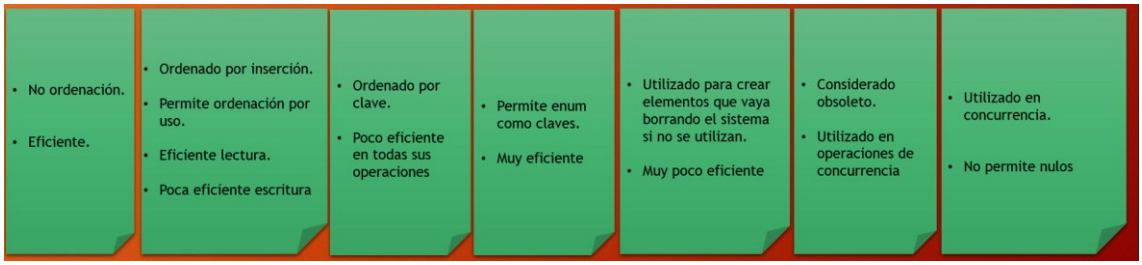
- Muy rápido al acceder al primer y último elemento
- Permite crear colas de elementos muy eficientes (LIFO/FIFO)

Las desventajas:

- Acceso lento a elementos intermedios

## Interfaz Map



## Iteradores

Dentro de java.util disponemos también de la interfaz Iterator que define objetos que permiten recorrer los elementos de una colección.

Los métodos disponibles dentro de esta interfaz son:

| Método | Uso |
|---|---|
| Object next() | Obtiene el siguiente objeto de la colección. Si h llegado al final genera una excepción NoSuchElementException |
| boolean hasNext() | Indica si hay un elemento siguiente (así evita excepción) |
| void remove() | Elimina el último elemento devuelto por next |

Ejemplo de recorrido de una colección:

```
/*suponiendo que coleccionString es una colección previamente creada donde se guardan
Strings */

Iterator it = coleccionString.itertor(); //se crea el objeto iterator sobre la colección

while (it.hasHext()){

  String elemento = (String) it.next(); //método next devuelve Object, necesario casting

  System.out.println (elemento);

}
```

Ejemplo 2 de recorrido de una colección usando for each

```
/*suponiendo que coleccionString es una colección previamente creada donde se guardan
Strings */

For(Object objetos: coleccionString){

  String elemento = (String) ojetos; // necesario casting

  System.out.println (elemento);

}
```