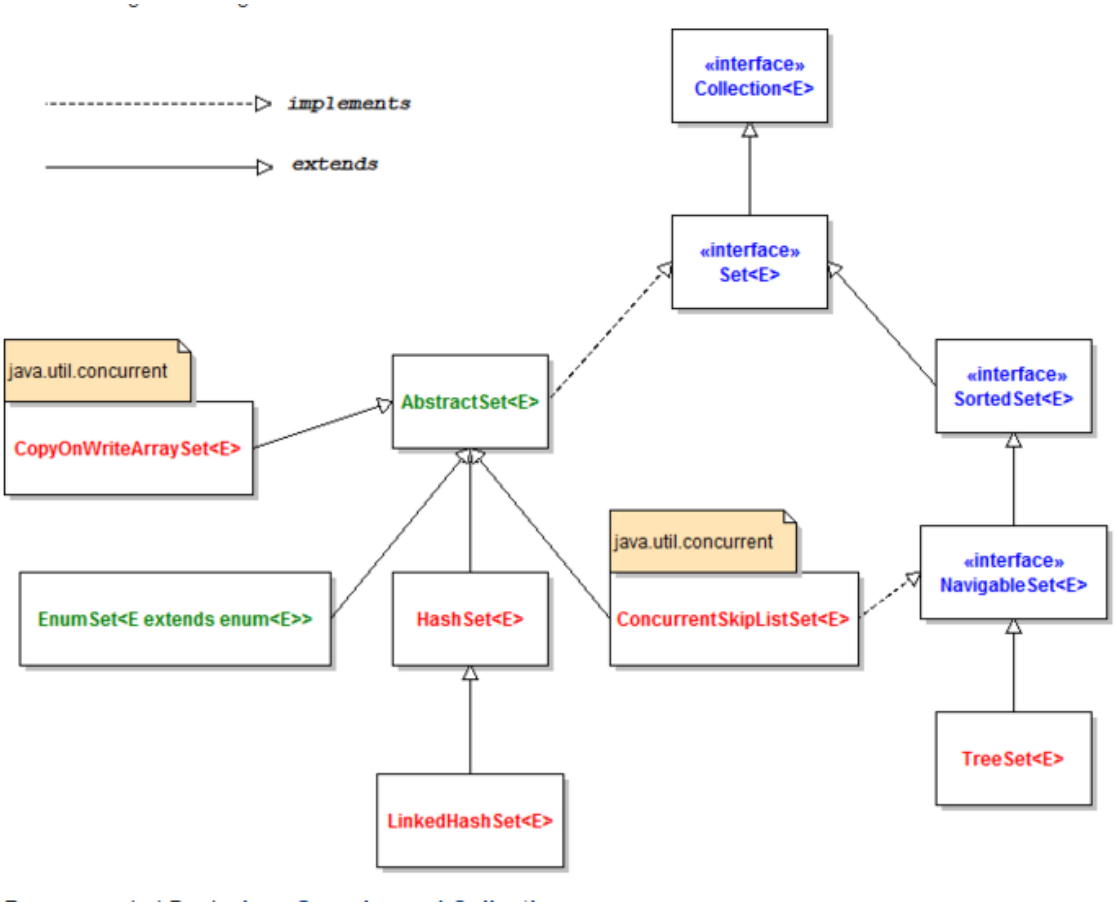
# ***EL INTERFACE SET***

En este documento veremos cuestiones sobre la colección SET y como ya vimos las interfaces SET, LIST Y QUEUE veremos una pincelada de un interface todavía más arriba en la jerarquía que es el interface Collection.

La diferencia conceptual entre un conjunto y una lista es que en un conjunto no hay orden ni elementos duplicados. También son diferentes las operaciones (métodos), por ejemplo, un conjunto tendrá implementado un método que permite la "intersección de conjuntos" que no es una operación característica de una lista.

***Perspectiva general***




HashSet, LinkedHashSet, TreeSet, ...todas las clases tendrán todos los métodos que indica la interface Set (size(), isEmpty(), ...). Como ya sabes, además cada clase concreta puede tener métodos adicionales e internamente están escritas con diferentes estrategias de almacenamiento. Nos fijamos en las siguientes clases

- HashSet -> almacenamiento basado en tabla hash

- LinkedHashSet -> almacenamiento basado en una mezcla de tabla hash y lista enlazada
- TreeSet -> almacenamiento basado en un árbol.

HashSet es el más eficiente para acceder aleatoriamente a sus elementos y por eso quizá es el más usado, no obstante, aunque un conjunto no tiene conceptualmente "orden", está claro que siempre existe subyacentemente un orden de almacenamiento y podemos querer usar ese orden para recorrer todo el conjunto, en ese caso nos puede interesar usar LinkedHashSet que almacena por orden de inserción y TreeSet que tiene sus elementos ordenados en un árbol como veremos más adelante en otro boletín.

**Ejemplo**: Crear un conjunto de Strings con la clase HashSet que almacena nombres de vehículos. Almacena "moto" "coche" y "bici".

```
import java.util.HashSet;

public class App {
    public static void main(String[] args) {
        HashSet<String> conjuntoVehiculos=new HashSet<>();
        conjuntoVehiculos.add("moto");
        conjuntoVehiculos.add("coche");
        conjuntoVehiculos.add("bici");
        conjuntoVehiculos.add("moto");
        conjuntoVehiculos.add("bici");

        for(String s: conjuntoVehiculos)
            System.out.print(s+" ");
    }
}
```

SALIDA:

```
moto coche bici
```

Observa que introdujimos dos veces moto y dos veces bici pero observamos que en la salida sólo hay una moto y una bici. Esto es debido a que por definición matemática de conjunto, un conjunto no puede tener elementos repetidos, y por esta razón, el add() de un HashSet no añade elementos repetidos. Aunque en el ejemplo anterior no utilizamos el valor de retorno de add(), éste devuelve un boolean, de forma que, si inserta el elemento devuelve true, false en caso contrario.

**Ejercicio U7_B5_E1:** Repite el ejemplo anterior pero utilizando la clase LinkedHashSet

**Ejercicio U7_B5_E2:** Repite el ejercicio anterior pero utilizando la clase TreeSet

Si comparamos la salida de los dos ejercicios anteriores vemos diferencias en el orden de impresión, esto es debido al orden de almacenamiento dentro del conjunto. Cuando el for recorre el conjunto va imprimiendo sus elementos según en el orden en que se los encuentra almacenados.
Con LinkedHashSet, observamos que internamente se almacenan por orden de inserción.
Con TreeSet, , observamos que internamente se almacenan en un árbol cuyo criterio de inserción es por orden alfabético

**Ejercicio U7_B5_E3:**
Modifica el código anterior para que la referencia conjuntoVehiculos sea de tipo Set.

**Ejercicio U7_B5_E4:**
Quizá las operaciones matemáticas más típicas de conjuntos son: union, diferencia e intersección. Haz un programa que prueba con los datos de los conjuntos A y B las tres operaciones indicadas. El resultado del programa debe ser el siguiente. Consulta el API para saber los métodos que debes usar para hacer las operaciones deseadas. Usa TreeSet ya que al ver los elementos ordenados es más fácil comprobar que las operaciones son OK.

```
run:
A:[5, 7, 9, 19]
B:[5, 7, 10, 20]
A union B: [5, 7, 9, 10, 19, 20]
A diferencia B: [9, 19]
A intersección B: [5, 7]
```

Si consultamos el interface Set concluimos que

A ∪ B => se consigue con addAll()
A-B => se consigue con removeAll()
A ∩ B => se consigue con retainAll()

**¿QUÉ SUBCLASE DE SET ELEGIR?**
Si el rendimiento es importante habría que estudiar mucho el tema. De forma superficial podemos quedarnos con:

- El más rápido para acceso directo es HashSet
- TreeSet más apropiado si quiero garantizar un orden
- Algo intermedio LinkedHashSet

En los siguientes retos deberás crear una solución basada en sets (pero puedes por supuesto adicionalmente usar otras estructuras si lo estimas necesario)

**Ejercicio U7_B5_E5: Reto** Michael J. Fox y el Pato Donald categoría conjuntos. Utiliza obligatoriamente conjuntos para resolverlo.

**Ejercicio U7_B5_E5:** RETO Bingo infantil 452. Puedes ayudarte de arrays pero no olvides incluir sets en tu solución.

## EL INTERFACE COLLECTION

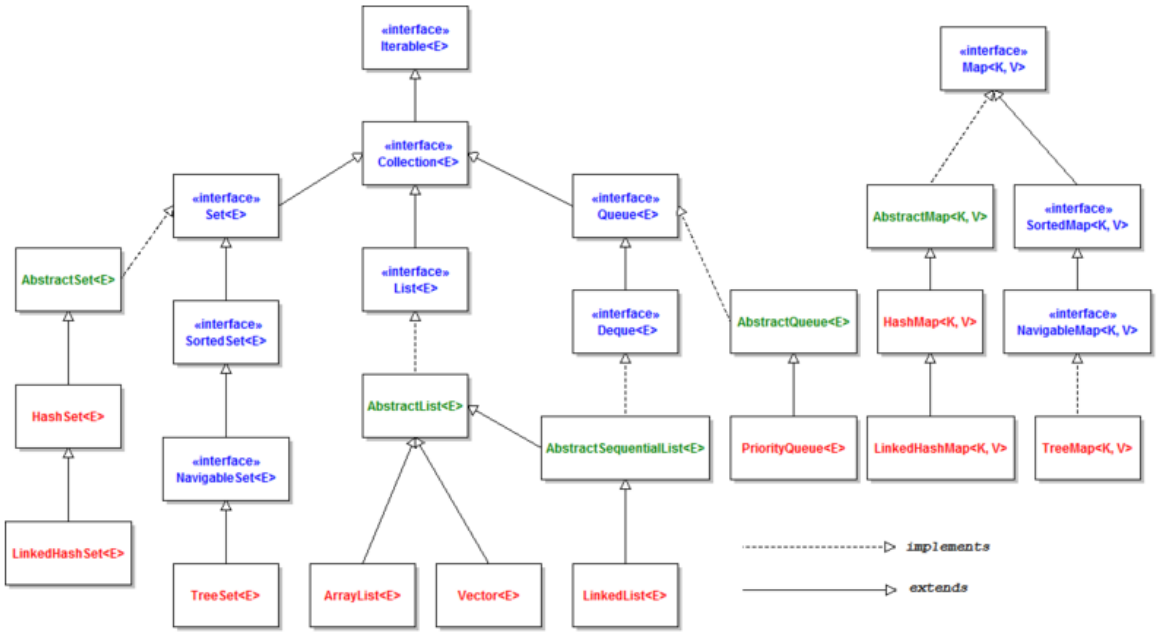
**Ejemplo trabajando al alto nivel Collection.**

Consulta el API de *Collection* y observa los métodos que define. Si el problema que tienes que resolver se puede resolver con los métodos especificados en Collection puedes hacer un código más abstracto, y por tanto mejor, trabajando a nivel Collection.

```
import java.util.*;
public class App{

  public static void main(String[] args) {
    // ArrayList
    Collection<String> al = new ArrayList<>();
    al.add("ArrayList1");
    al.add("ArrayList2");
    al.add("ArrayList3");
    for(String s:al)
       System.out.print(s+" ");
    System.out.println();
    // LinkedList
    Collection<String> ll = new LinkedList<>();
    ll.add("LinkedList1");
    ll.add("LinkedList2");
    ll.add("LinkedList3");
    for(String s:ll)
       System.out.print(s+" ");
    System.out.println();
    // HashSetC
    Collection<String> hs = new HashSet<>();
    hs.add("hs1");
    hs.add("hs2");
    hs.add("hs2");
    hs.add("hs3");
    for(String s:hs)
       System.out.print(s+" ");
    System.out.println();

    // ¡ERROR! HashMap no es una Collection
    //Collection hm = new HashMap();

  }
}
```

Observa que el segundo hs.add("hs2"), como se ejecuta sobre hashSet ya que el add que se ejecuta es el sobrescrito en la clase hashmap no se añade. Recuerda que en un hashset no puede haber elementos repetidos

Después de haber trabajado con clases concretas relativas al procesamiento de colecciones, estamos en disposición de entender un poco mejor el siguiente gráfico que usamos anteriormente para explicar a que se refiere el término "FrameWork Collections".




Class diagram of Java Collections framework

Hacemos las siguientes observaciones:

- Dentro del FrameWork Collections hay como dos ramas independientes: lo que cuelga del interface Collection y lo que cuelga del interface Map.
- Todo lo que no es mapa, al más alto nivel es una Collection.

Ahora dentro de el "árbol" que cuelga de Collection simplificando mucho podemos distinguir dos subramas:

- el interface Set
- el interface List

**OJO CON LA PALABRA COLLECTION**

Se emplea para cosas distintas en el API:

- Hablamos del FrameWork Collections
- Hablamos del Interface Collection que acabamos de estudiar más arriba
- Hablamos de la clase Collections que es un conjunto de métodos static, utilidades para trabajar con colecciones. Iran apareciendo poco a poco en los ejemplos alguno de estos métodos.