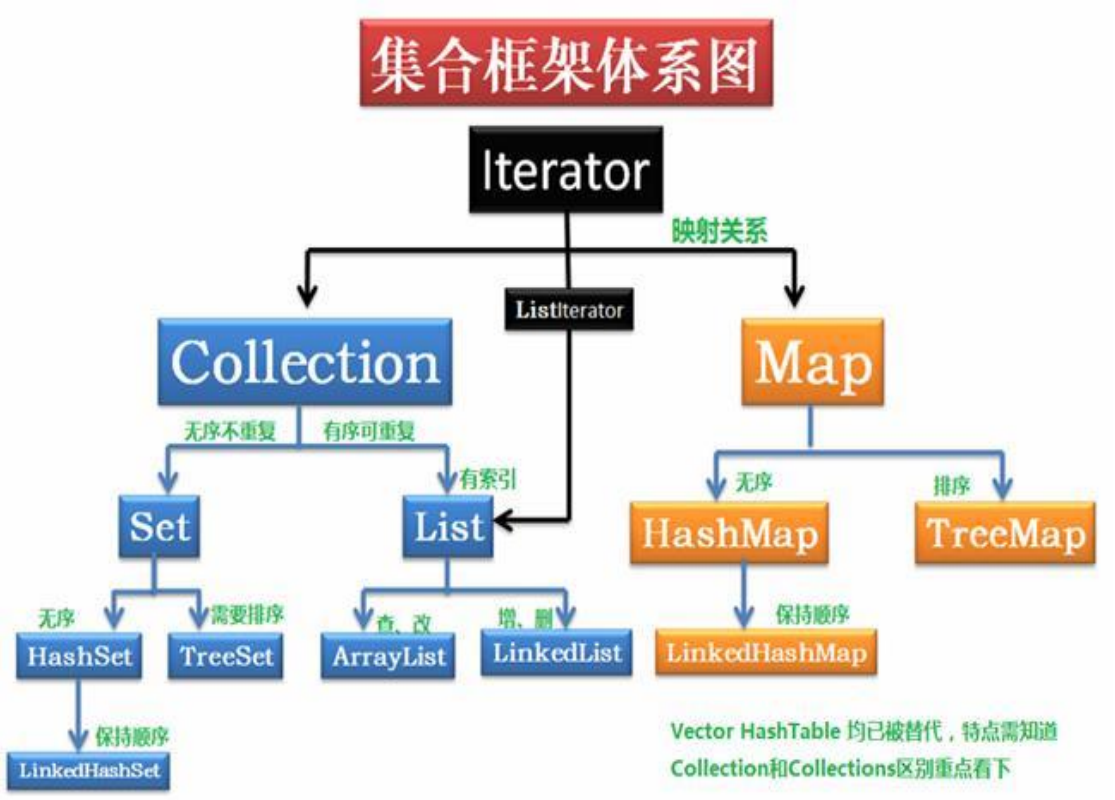
En los lenguajes de programación suele haber dos tipos de datos almacenados que son:

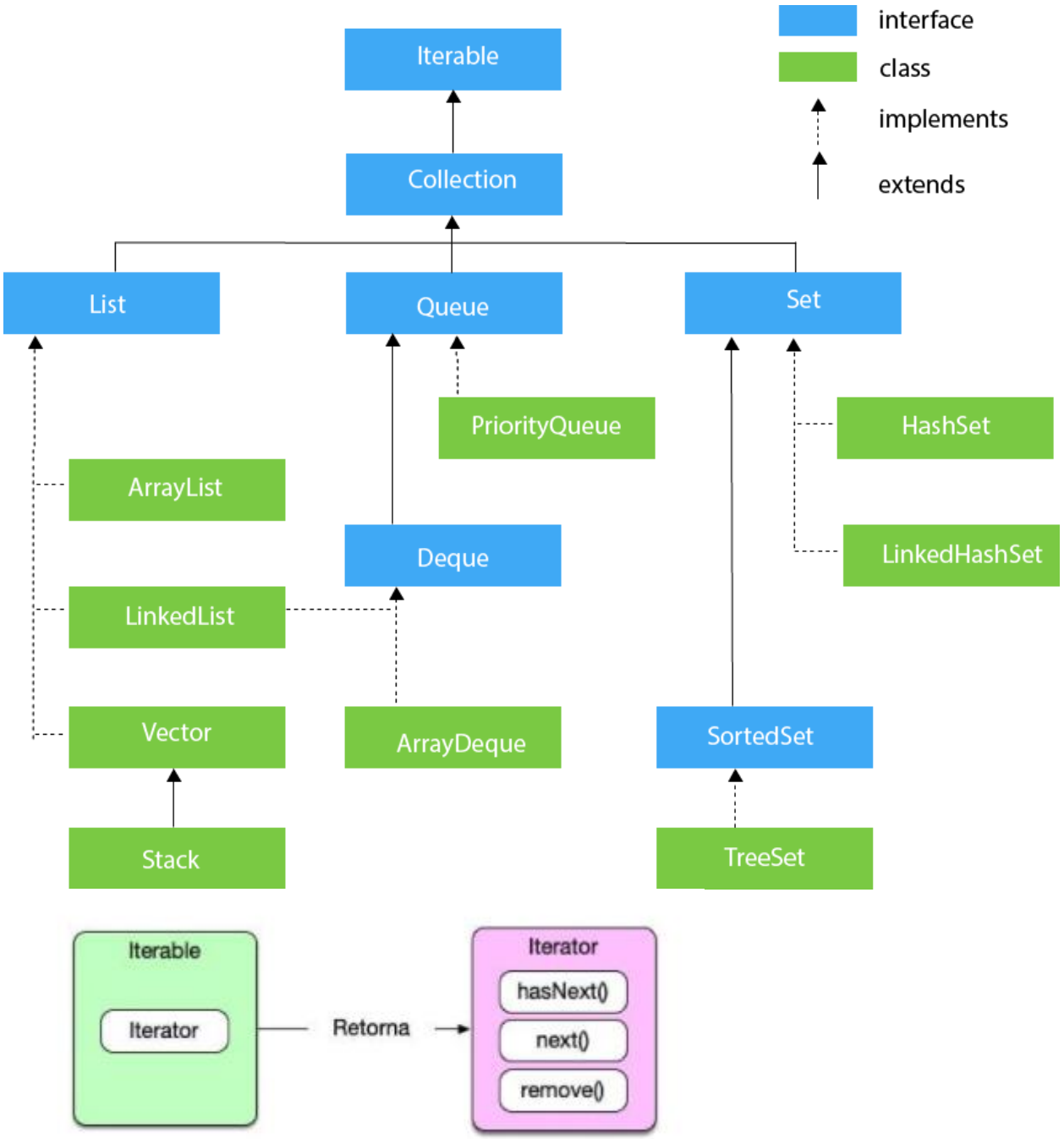
- Estáticos. Son los arrays que ya has manejado, debemos saber de antemano el número de elementos con los que deseamos trabajar.
- Dinámicos. No sabemos cuántos elementos vamos a tener y según necesitamos elementos los vamos incorporando. Son los que están agrupados en las colecciones o Collection. Que son las listas, pilas, colas, árboles, grafos, etc…

Los datos que forman las colecciones de datos dinámicos son siempre objetos y cada uno de esos objetos recibe el nombre de nodo. Esos objetos tienen un campo llamado campo de enlace, cuyo tipo es el nombre de la clase donde está y se utiliza para enlazar con otro elemento. Estas estructuras pueden ser:

1. Lineales. Listas, pilas y colas.
2. No lineales. Árboles y grafos.

Tenemos una serie de clases e interfaces para manipular estos tipos de estructuras. Que son Collection, Map y Iterator.









INTERFAZ COLLECTION

Se encuentra en **java.io.Collection**. Solo se pueden recorrer los datos en el mismo orden en que fueron insertados y debemos pasar por todos los anteriores hasta el elemento deseado. Podemos eliminar cualquier elemento de la colección. Y los elementos nuevos van al final de la colección.

Tenemos una serie de métodos que son comunes para las colecciones, salvo alguna excepción. La lista de métodos es:

- **boolean add(Object o)**. Añade el objeto a la colección. Devuelve true si se pudo completar la operación. Si no cambió la colección como resultado de la operación devuelve false
- **boolean remove(Object o)**. Elimina al objeto indicado de la colección.

- **int size( )**. Devuelve el número de objetos almacenados en la colección.
- **boolean isEmpty( )**. Indica si la colección está vacía.
- **boolean contains(Object o)**. Devuelve true si la colección contiene a o.
- **void clear( )**. Elimina todos los elementos de la colección.
- **boolean addAll( Collection otra)**. Añade todos los elementos de la colección otra a la colección actual.
- **boolean removeAll( Collection otra)**. Elimina todos los objetos de la colección actual que estén en la colección otra.
- **boolean retainAll(Collection otra)**. Elimina todos los elementos de la colección que no estén en la otra.
- **boolean containsAll(Collection otra)**. Indica si la colección contiene todos los elementos de otra
- **Object[] toArray( )**. Convierte la colección en un array de objetos.
- **Iterator iterator( )**. Obtiene el objeto iterador de la colección.

La interfaz Iterator (también en java.util) define objetos que permiten recorrer los elementos de una colección. Los métodos definidos por esta interfaz son:

- **Object next( )**. Obtiene el siguiente objeto de la colección. Si se ha llegado al final de la colección y se intenta seguir, da lugar a una excepción de tipo: **NoSuchElementException** (que deriva a su vez de **RunTimeException**)
- **boolean hasNext( )**. Indica si hay un elemento siguiente (y así evita la excepción).
- **void remove( )**. Elimina el último elemento devuelto por **next**.

Para recorrer los elementos de una lista podemos utilizar el iterador Iterator o la interfaz ListIterator, que nos da mayores posibilidades, al poder modificar, añadir objetos y recorrer la lista en ambos sentidos.

Sus métodos son:

- **int nextIndex( ),** devuelve la posición del siguiente elemento.
- **int previousIndex( ),** devuelve la posición del objeto al que apunta.
- **Object previous( ),** devuelve el objeto al que apunta.

Ejemplo:

```
public static void main(String [] args)
{
  //Collection  es un  interfaz y no puede instanciarse
  Collection  elemento = new ArrayList();
  elemento.add(25);
  elemento.add(54);
  elemento.add(23);
  int valor;
  Iterator  nodo = elemento.iterator();
  while( nodo.hasNext())
 {
    valor = (Integer)elemento.next( );
    System.out.println(valor+"  ");
 }
}
```

Podemos tener las interfaces List y Set.

a) **Interfaz List**. Es una colección de nodos que están en cierto orden, guarda la información de cómo están colocados cosa que no hace Collection, permite duplicados y permite el acceso a los datos a partir de una posición.

La interfaz List añade nuevos métodos para trabajar como pueden ser añadir un elemento entre dos nodos, mostrar un elemento indicando su posición (se enumeran de 0 en adelante) que son:

**int lastIndexOf(Object elemento)** Devuelve la posición del elemento comenzando a buscarle por el final. Si no lo encuentra, devuelve -1.
**void addAll(int índice, Collection elemento)** Añade todos los elementos de una colección a una posición dada.
**ListIterator listIterator( )** Obtiene el iterador de lista que permite recorrer los elementos de la lista.
**ListIterator listIterator(int índice)** Obtiene el iterador de lista que permite recorrer los elementos de la lista. El iterador se coloca inicialmente apuntando al elemento cuyo índice en la colección es el indicado.

Si intentamos acceder a una posición que no existe se produce la excepción IndexOutOfBoundsException.

Esta implementada por dos clases: ArrayList y LinkedList.

▪ **ArrayList**, no tiene un tamaño fijo de nodos o elementos. Indicamos su posición y accedemos al elemento directamente. Se pierde tiempo al borrar un elemento ya que tiene que recolocar todos los elementos que van detrás del elemento borrado al igual que insertamos un elemento.

**Ejemplos:**
**ArrayListEjemplo.java**
**ArrayListEjemplo_String.java**

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

▪ **Stack**, es una pila que los elementos se accede por el método LIFO (Last In First Out). Sus accesos son sincronizados. Podemos decir que tenemos métodos propios para su uso.

**Boolean empty (),** devuelve true si la pila esta vacía.
**Objeto peek( ),** devuelve el elemento situado en la cima de la pila.
**Objeto pop(),** saca el elemento de la pila que está arriba.
**Ojbeto pusp(objeto valor),** añade un nuevo elemento a la pila.
**Int search(Object valor**), devuelve la distancia del valor buscado con la cima de la pila. Si no encuentra el objeto devuelve -1.

**Ejemplos:**
**StackEjemplo.java**

https://docs.oracle.com/javase/7/docs/api/java/util/Stack.html
https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=71&codigo=149&inicio=60

- **LinkeList**, son listas doblemente enlazadas, cada nodo o elemento tiene dos campos de enlace para saber quién le precede y quien le antecede. Se insertan o borran elementos de forma más fácil, al solo cambiar las referencias de los enlaces.

Algunos de los métodos propios son:

**addFirst(obj valor), offerFirst(obj valor) y push(obj valor)**, inserta un elemento al comienzo de la lista.
**addLast(obj valor), offer(obj valor) y offerLast(obj valor)**, inserta un elemento al final de la lista.
element**(), gerFirst(), peek() peekFirst(),** devuelve el elemento situado al comienzo de la lista.
**Poll(), pollFirst(), pop(), remove(), removeFirst()**, devuelve y elimina el elemento situado al comienzo de la lista.
**getLast() y peekLast()**, devuelve ele elemento situado al final de la lista.
**pollLast() y removeLast()**, elimina el elemento situado al final de la lista.

**Iterator <Objeto> descendingIterator()**, permite recorre la lista en orden inverso.
**boolean removeFisrtOccurrence(obj valor)**, elimina la primera ocurrencia que coincide con el valor pasado.
**Boolean removeLastOcurrence(obj valor)**, elimina la última ocurrencia del valor pasado.

https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html
https://www.redeszone.net/2012/03/05/curso-de-java-estructuras-de-datos-arraylist-y-linkedlist/
https://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?punto=73&codigo=152&inicio=60

b) Interfaz Set. Se usa cuando queremos que en la colección no tengamos elementos repetidos en la colección. Esta implementada por dos clases: HashSet y TreeSet.

- **HashSet,** es la clase más utilizada para implementar una colección sin duplicados, pero no se garantizan su orden. Utiliza una tabla, guardando los diferentes códigos de identificación de la colección. Cada posición de la tabla apunta a los diferentes nodos que tienen el mismo código de identificación. Debemos sobreescribir los métodos equals (decidimos cuando son iguales dos objetos) y hashCode (determinamos cual es el código de identificación de cada objeto). No admite acceso secuencial.

  **HashSet  nombre = new HashSet( );**

  Ejemplo:  **HashSet <String> palabra = new HashSet<String>();**

**Ejemplos:**
**HastSetEjemplos.java**
**HastSetEjemplos_String.java**

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/HashSet.html

- **LinkedHashSet,** no guardar duplicados, pero si mantiene el orden de los valores introducidos.

  **LinkedHashSet nombre = new LinkedHashSet();**

  Ejemplo: **LinkedHashSet <Integer> numeros = new LinkedHashSet<>();**

  **Ejemplos:**
  **LinkedHastSetEjemplos.java**
  **LinkedHastSetEjemplos_String.java**

  https://docs.oracle.com/javase/10/docs/api/java/util/LinkedHashSet.html

- **TreeSet**, implementa la interfaz SortedSet, los elementos se encuentran ordenados en estructura en árbol binario. Con este tipo debemos implementar Comparable a la clase.

  **TreeSet nombre = new TreeSet( );**

  **Ejemplos:**
  **TreeSetEjemplos.java**
  **TreeSetEjemplos_String.java**

  Si los tipos de datos tienen una ordenación diferente a la esperada debemos utilizar lo siguiente:

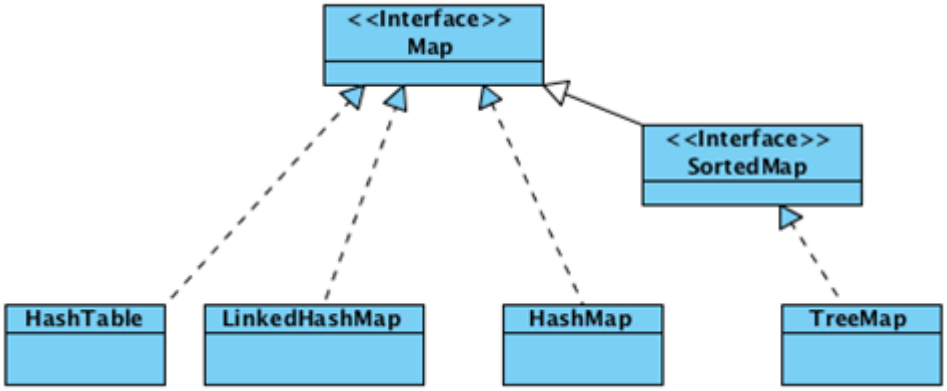
  **public class nombre implements Comparable <nombre clase>**

  Comparable es un interfaz y tenemos que recodificar el método public int compareTo(Objeto). Si devuelve positivo el objeto que le llama va después que el objeto que recibe, 0 si son iguales y negativo el objeto que le llama va antes que el objeto que recibe.

  https://docs.oracle.com/javase/10/docs/api/java/util/TreeSet.html

INTERFAZ MAP

Los mapas guardan parejas de objetos, asociando un objeto por su clave con otro objeto por un valor concreto. Las claves no pueden estar repetidas y solo se pueden asociar con un valor.




Tenemos varios tipos de Map que son:

a) **HashMap**, almacena las claves en una tabla hash y admite valores nulos, ni valores duplicados. No garantiza ningún orden.

**HashMap<> nombre = new HasMap<>();**

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
https://www.w3schools.com/java/java_hashmap.asp
**https://beginnersbook.com/2013/12/hashmap-in-java-with-example/**

b) **LinkedHashMap**, tiene un enlace doble con los elementos. Mantiene el orden de entrada.

**LinkedHashMap<> nombre = new LinkedHasMap<>();**

https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html
**https://www.callicoder.com/java-linkedhashmap/**

c) **TreeMap**, almacena las claves ordenadas. No admite valores nulos. Tiene métodos propios.

**TreeMap<> nombre = new TreeMap<>();**

https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html
**https://beginnersbook.com/2013/12/treemap-in-java-with-example/**

**Map< > nombre = new HasMap<>();**

```
// Declaración de un Map (un HashMap) con clave "Integer" y Valor "String". Las claves pueden ser de cualquier tipo de objetos, aunque los más utilizados como clave son los objetos predefinidos de Java como String, Integer, Double ... !!!!CUIDADO los Map no permiten datos atómicos

Map<Integer, String> nombreMap = new HashMap<Integer, String>();
```

Los métodos que podemos utilizar con MAP son:

**size()**, nos devuelve el número de elementos de un Map.
**isEmpty()**, devuelve true si no hay elementos en el Map y false si los hay.
**put(elemento)**, añade un elemento al Map.
**get(elemento)**, devuelve el valor de la clave que se le pasa como parámetro o null si la clave no existe.
**clear()**, borra todos los elementos del Map.
**remove(elemento)**, borra el elemento pasado.
**containsKey(elemento)**, devuelve true si en el Map hay una clave que coincide con el valor pasado.
**containsValue(elemento)**, devuelve true si en el Map hay un valor que coincide con el pasado.
**values()**, devuelve la colección de los valores de Map.

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html
https://jarroba.com/map-en-java-con-ejemplos/
https://pablomonteserin.com/curso/java/java-map/

https://www.arquitecturajava.com/java-foreach-y-sus-opciones/