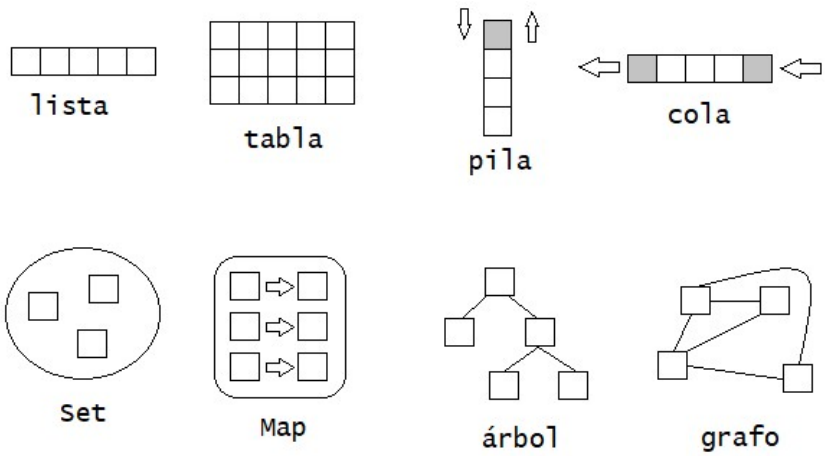
Java Collection Framework

El Java Collection Framework es una librería que sirve para crear distintos tipos de **colecciones de objetos**[1].

> Una **colección de objetos** es un objeto que guarda otros objetos.

En general, en informática se utilizan varios tipos de colecciones de objetos, como estas:



- **Lista**: Es una secuencia ordenada de datos, en los que cada uno ocupa una posición. Por ejemplo, la lista de alumnos de una clase, por orden de lista.
- **Tabla**: Es una disposición de datos en filas y columnas. Por ejemplo, el horario de clases de un grupo de alumnos es una tabla.
- **Pila**: Estructura que se forma cuando los datos se añaden uno encima de otro, de forma que siempre tenemos acceso al que está encima de todos. Por ejemplo, una baraja de cartas en la que sacamos la de arriba es una pila.
- **Cola**: Estructura en la que los datos se añaden uno a continuación de otro, de forma que el primero en salir de la cola es el primero que llegó. Por ejemplo, las personas que esperan para entrar en el cine forman una cola.
- **Set**: Colección de objetos diferentes en la que no se tiene en cuenta el orden. Por ejemplo, los alumnos que están en el patio del recreo forman un Set.
- **Map**: Es una estructura que almacena parejas de datos asociados. Por ejemplo, las asociaciones entre el nombre de usuario y su contraseñan forman un Map.
- **Árbol**: Estructura en la que hay un dato padre (raíz) del que cuelgan datos hijos. A su vez, cada hijo puede tener varios hijos y así sucesivamente. Por ejemplo, la estructura de carpetas y subcarpetas de un disco duro es un árbol.
- **Grafo**: Estructura en la que cualquier dato puede estar relacionado con los demás sin ningún tipo de restricción. Por ejemplo, relacionar una persona con todas las personas que conoce formaría un grafo.

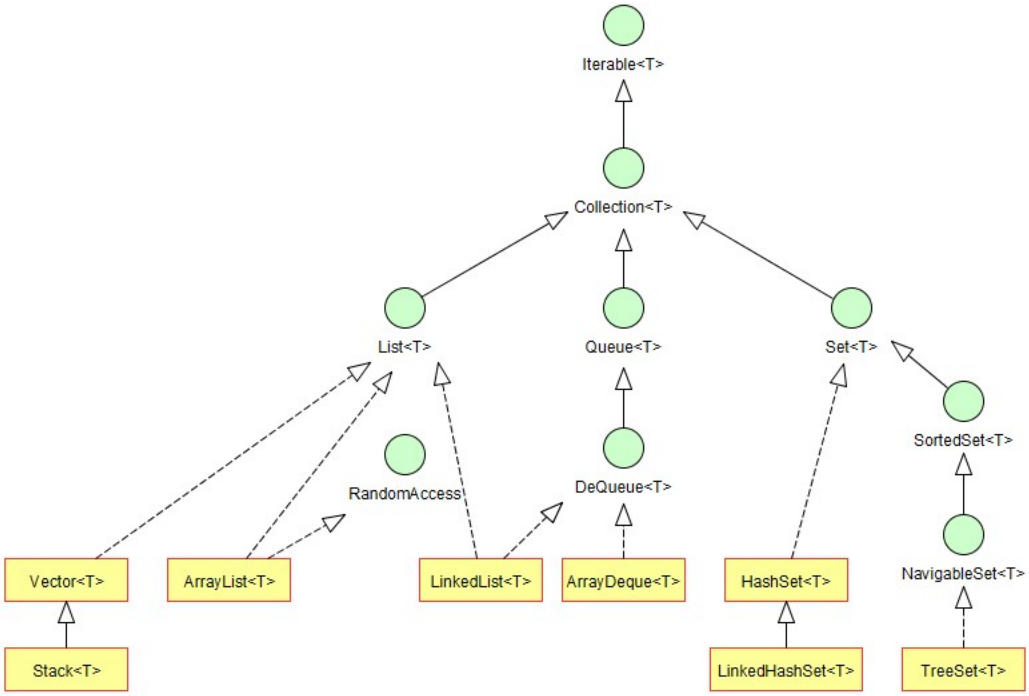
[1] Las colecciones de objetos también se denominan **estructuras de datos**.

El **Java Collection Framework** es una librería de clases e interfaces incluida en el paquete **java.util** que nos permite crear estas colecciones:

- **Listas** → se usará la interfaz **List<T>** y todas las clases que la implementan, siendo la más habitual **ArrayList<T>** (T es el tipo de objeto que se guarda)
- **Pilas y Colas** → Se usarán las interfaces **Queue<T>** y **DeQue<T>**, siendo la más implementación más habitual **ArrayDeque<T>**
- **Set** → Se usará la interfaz **Set<T>**, siendo **HashSet<T>** su implementación más habitual
- **Map** → Se usará la interfaz **Map<K,V>**, siendo la implementación más habitual la clase **HashMap<K,V>**

El diagrama de clases del Java Collection Framework tiene dos partes.

La primera está formada por las clases e interfaces para trabajar con Listas, Pilas, Colas y Set:

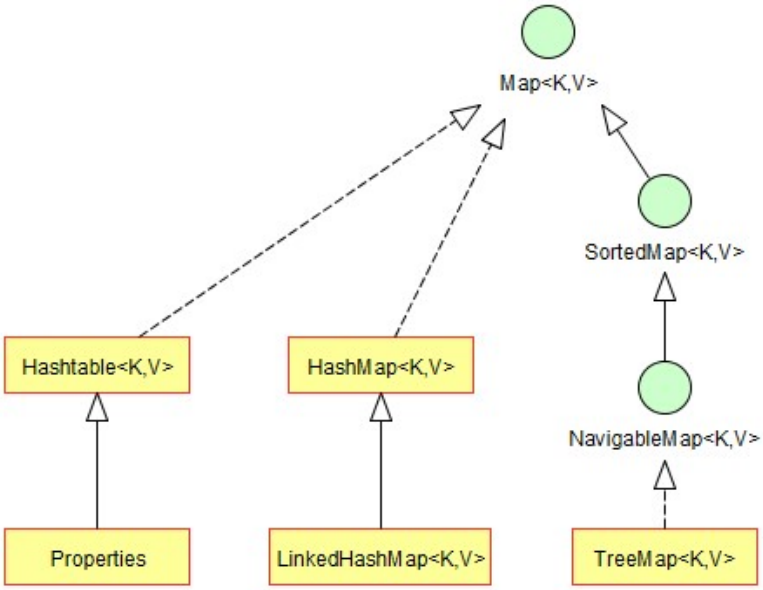


Como podemos ver, todo comienza en la interfaz **Iterable<T>**, que representa una colección cuyo contenido puede ser recorrido con un **for mejorado**.

A partir de ahí, **Collection<T>** es la interfaz padre de todas las colecciones de objetos y posee los métodos comunes a las listas, pilas, colas y Set.

Por último, tenemos las interfaces **List<T>, Queue<T>,DeQue<T>** y **Set<T>** que representan las listas, colas, pilas y set respectivamente. Cada una posee los métodos que las diferentes clases implementadoras recibirán.

La segunda parte del Java Collection Framework permite trabajar con Map:



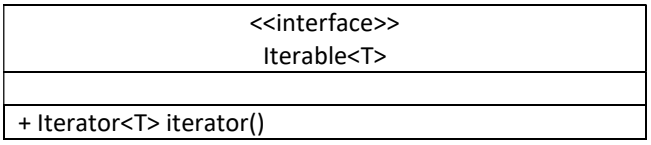
La interfaz **Map<K,V>** representa un Map y luego tenemos distintas clases que la implementan y otras interfaces que heredan de ella.

En la documentación adjunta describimos los principales métodos de cada una de las clases e interfaces del Java Collection Framework, así como unas clases auxiliares como **Collections** o **Arrays** que nos facilitan algunas cosas.

1) Iterable<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Dado un tipo de dato referencia T, esta interfaz representa una colección de objetos de tipo T que puede ser recorrida hacia adelante con un for mejorado.



- **iterator**: Permite obtener un objeto para recorrer la colección con un bucle while. Este enfoque está obsoleto y no se recomienda usar este método. En su lugar, se prefiere un for mejorado para recorrer el objeto.

2) Collection<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Dado un tipo de dato referencia T, esta interfaz representa una colección de objetos genérica que guarda objetos de tipo T.

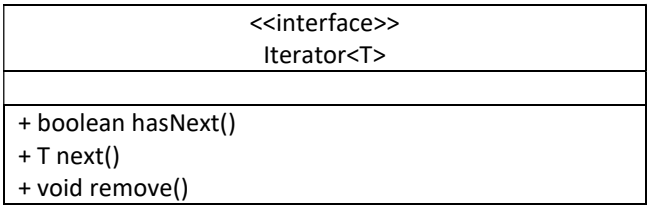
<<interface>> Collection<T>
+ boolean add(T obj) + boolean addAll(Collection<? extends T> c) + boolean clear() + boolean contains(Object o) + boolean containsAll(Collection<?> c) + boolean isEmpty() + Iterator<T> iterator() + boolean remove(Object o) + boolean removeAll(Collection<?> c) + int size()

- **add**: método que añade un objeto en la colección y devuelve true si ha sido posible hacerlo.
- **addAll**: método que añade al objeto todos los objetos de la colección pasada como parámetro.
- **clear**: borra todos los objetos de la colección
- **contains**: devuelve true si el objeto pasado como parámetro está en la colección. Para comparar los objetos se utiliza el método equals que heredan de la clase Object.
- **containsAll**: devuelve true si el objeto contiene a todos los elementos de la colección pasada como parámetro.
- **isEmpty**: devuelve true si la colección está vacía.
- **iterator**: método que devuelve un objeto con el que se pueden recorrer todos los elementos de la colección mediante un bucle while.
- **remove**: elimina de la colección el objeto pasado como parámetro, devolviendo true si se borra con éxito
- **removeAll**: elimina del objeto todos los elementos de la colección pasada como parámetro.
- **size**: devuelve el número de elementos que hay en la colección.

3) Iterator<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Este objeto se obtiene llamando al método iterator() de la interfaz Iterable<T> y sirve para recorrer (visitar) todos los objetos de la colección usando un bucle while. Ya no se recomienda su uso porque hoy tenemos el for mejorado, con el que podemos hacer lo mismo mucho más fácilmente.

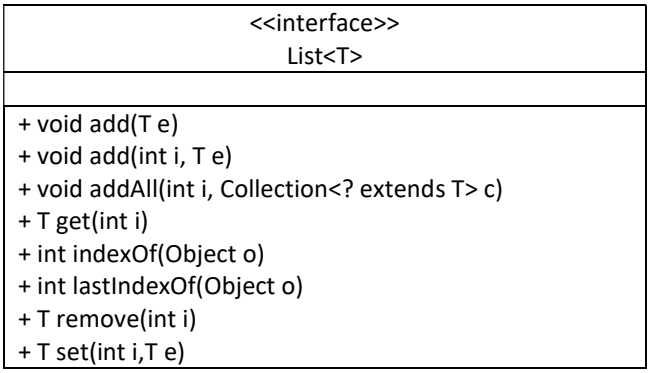


- **hasNext**: método que devuelve true si quedan elementos de la colección por recorrer.
- **next:** método que devuelve el siguiente objeto de la colección que está siendo recorrida por el iterador.
- **remove:** método que borra de la colección el último elemento que ha sido devuelto por el método next. Es posible que este método no funcione en algunas colecciones.

4) List<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Dado un tipo de dato referencia T, esta interfaz representa una lista de objetos de tipo T. Por tanto, es una colección en la que los objetos tienen una posición en la lista, que empieza por cero.



- **primer add**: método que añade un objeto al final de la lista.
- **segundo add**: método que inserta un objeto en la posición indicada de la lista, desplazando todos los demás una posición hacia la derecha.
- **addAll**: método que inserta en la posición indicada de la lista todos los objetos de la colección recibida como parámetro.
- **get**: devuelve el objeto que se encuentra en la posición indicada como parámetro.
- **indexOf**: devuelve la posición de la primera aparición de un objeto en la lista. Para comparar los objetos hasta encontrar el buscado se utiliza el método equals.
- **lastIndexOf**: devuelve la posición de la última aparición de un objeto en la lista.
- **remove**: borra de la lista el objeto cuya posición se pasa como parámetro. Devuelve el objeto que ha sido borrado.
- **set**: reemplaza el objeto que está en la posición indicada en el primer parámetro por el objeto recibido como segundo parámetro.

5) ArrayList<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Esta clase representa una lista de tamaño variable, a la que se pueden añadir objetos de forma indefinida, hasta que se acabe la memoria del equipo.

ArrayList<T>
+ ArrayList<T>() + ArrayList<T>(Collection<? extends T> c)

- **primer constructor:** Crea una lista vacía.
- **segundo constructor:** Crea una lista que contiene todos los objetos de la colección pasada como parámetro.

6) Collections

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Esta clase proporciona utilidades que facilitan el trabajo con las colecciones.

Collections
+ static void shuffle(List<T> lista) + static void sort(List<T> lista) + static int binarySearch(List<T> lista, T objeto)

- **shuffle:** Desordena de forma aleatoria todos los objetos de la lista
- **sort:** Ordena la lista recibida como parámetro, de menor a mayor, siempre que T sea un tipo de dato que pueda ser ordenado (Comparable).
- **binarySearch:** Devuelve la posición en la que se encuentra un objeto en la lista, usando un algoritmo llamado "búsqueda binaria", que tiene coste logarítmico (*tener en cuenta que el método indexOf usa coste lineal, por lo que es menos eficiente*). Sin embargo, para que este método funcione la lista pasada como parámetro debe haber sido previamente ordenada.

7) Arrays

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Esta clase proporciona utilidades que facilitan el trabajo con los arrays.

Collections
+ static List<T> asList(T… datos)

- **asList:** Devuelve una lista con todos los elementos que se pasan como parámetros. La lista obtenida no es un ArrayList, sino que es una lista de solo lectura en la que solo se pueden leer los elementos.

8) Set<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

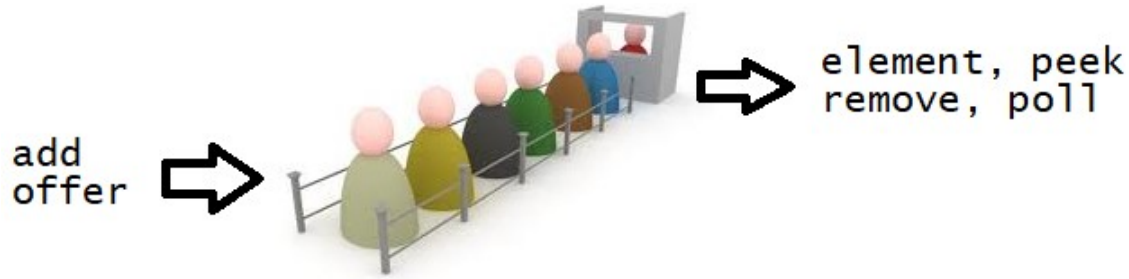
Esta interfaz representa un conjunto de objetos en los que en principio no existe ninguna relación de orden (es decir, no hay posición), y **no se admiten elementos duplicados**.

No tiene métodos nuevos distintos a los que recibe de Collection<T>.

9) Queue<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Esta interfaz hereda de Collection y representa una **cola** que contiene objetos de tipo T. Es decir, es una estructura de datos en la que los elementos se insertan por el final, pero se recuperan por el principio[2] como las colas de la vida real.



Queue<T>
+ boolean add(T obj) + boolean offer(T obj) + T element() + T peek() + T remove() + T poll()

- **add:** Añade un elemento al final de la cola y devuelve true si puede ser añadido. En algunos tipos de colas es posible que no haya espacio para añadir el objeto y entonces se lanzará una IllegalStateException.
- **offer:** Es como el método add, pero en caso de que no haya espacio suficiente para añadir el objeto, devuelve false en lugar de lanzar una excepción.
- **element:** Devuelve el elemento que está en la primera posición de la cola. En caso de que la cola esté vacía lanza una NoSuchElementException
- **peek:** Es como el método element, pero en caso de que la cola esté vacía devuelve null y no lanza ninguna excepción.
- **remove:** Devuelve el elemento que está en la primera posición de la cola, y además lo elimina de ella. En caso de que la cola esté vacía lanza una NoSuchElementException
- **poll:** Es como remove, pero si la cola está vacía devuelve null.

[2] Esta forma de funcionar se denomina FIFO (First In, First Out): el primero que llega es el primero en salir.

10) ArrayDeque<T>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Esta clase es una implementación de Deque<T> (en particular, es una Queue<T>), y puede ser usada como cola (tiene los métodos de Queue<T>) y también como pila (tiene los métodos definidos en Deque<T>).

Esta colección no permite guardar en ella el valor **null**. Si se intenta, se lanzará una NullPointerException.

ArrayDeque<T>
+ ArrayDeque() + ArrayDeque(Collection<? Extends E> c)

- El primer constructor crea una ArrayDeque<T> vacía
- El segundo constructor crea una ArrayDeque<T> que contiene los objetos de la colección pasada como argumento.

11) Map<K,V>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Representa una colección de asociaciones entre dos datos (por ejemplo, nombre de usuario y contraseña). El dato principal, que debe ser único, se llama clave (su tipo es K) y el otro se denomina valor (su tipo es V).

Map<K,V>
+ V put(K k,V v) + V putIfAbsent(K k, V v) + V get(Object o) + V getOrDefault(Object o, V v) + boolean isEmpty() + int size() + V remove(Object o) + boolean remove(Object k, Object v) + void clear() + boolean containsKey(Object o) + boolean containsValue(V v) + Set<K> keySet() + Collection<V> values()

- **put:** Añade al Map la asociación entre la clave y valor pasados como parámetro. Si la clave ya está en el Map, actualiza su valor. El método devuelve el valor previo que estuviese asociado a la clave, o null si no había ninguno.
- **putIfAbsent:** Añade al Map la asociación entre la clave y el valor pasados como parámetro solo si la clave no está en el Map (o está en el Map, pero su valor asociado es null)
- **get:** Devuelve el valor correspondiente a la clave que se pasa por parámetro, o null si dicha clave no se encuentra en el Map.
- **getOrDefault:** Devuelve el valor al que está asociada la clave. En caso de que la clave no se encuentre en el Map, devuelve el valor que se indica como segundo parámetro.
- **isEmpty:** Devuelve true si el Map está vacío
- **size:** Devuelve el número de asociaciones que guarda el Map.
- **Primer remove:** Borra del Map la clave que se pasa como parámetro y su valor. Devuelve el valor asociado a la clave que ha sido borrada.
- **Segundo remove:** Borra del Map la clave pasada como parámetro solo si está asociada al valor pasado como segundo parámetro.
- **clear:** Borra todas las claves y sus valores asociados
- **containsKey:** Devuelve true si el objeto pasado como parámetro es una clave del Map
- **containsValue:** Devuelve true si el objeto pasado como parámetro es un valor del Map
- **keySet:** Devuelve un Set rellenado con todas las claves que hay en el Map
- **values:** Devuelve un Collection rellenado con todos los valores que hay en el Map

12) HashMap<K,V>

- **Librería:** *Java Collection Framework*
- **Archivo:** *Incluido en el JDK*
- **Desarrollador:** *Sun Microsystems*
- **Paquete:** *java.util*

Es una clase que implementa la interfaz Map<K,V> y por tanto, tiene todos sus métodos.

HashMap<K,V>
+ HashMap()

- **constructor:** Crea un HashMap vacío, listo para irle añadiendo parejas (clave,valor)