Las colecciones en Java

El API de Java nos proporciona el framework de las colecciones, que nos permite utilizar diferentes estructuras de datos para almacenar y recuperar objetos de cualquier clase. Dichas colecciones no forman parte del lenguaje, sino que son clases definidas en el paquete `java.util`. Para crear una colección usaremos la siguiente estructura:

```
Coleccion<Clase> nombre = new Coleccion<Clase>();
```

Donde `Coleccion` es una clase de este framework que queramos utilizar según la estructura de almacenamiento que nos interese y `Clase`representa el tipo de datos a almacenar. Por ejemplo, para crear una lista ordenada de objetos de la clase `String` escribiríamos:

```
ArrayList<String> listaNombres =  new ArrayList<String>();
```

En Java podemos trabajar con clases genéricas, para ello se utiliza <y >. A este mecanismo se le conoce como genericidad.

Hay tres tipos de colecciones, cada uno con un interfaz común y diferentes implementaciones. Las diferentes implementaciones de un mismo interfaz realizan la misma tarea aunque la diferencia está en que unas implementaciones son más rápidas en algunas operaciones y más lentas en otras:

Conjunto – los elementos no tienen un orden y no se permiten duplicados. Se define el interfaz `Set<E>`. Podemos utilizar las siguientes implementaciones:

- `HashSet<E>` (implementación con tabla hash)
- `LinkedHashSet<E>` (tabla hash +doble lista enlazada)
- `TreeSet<E>` (implementación con árbol)

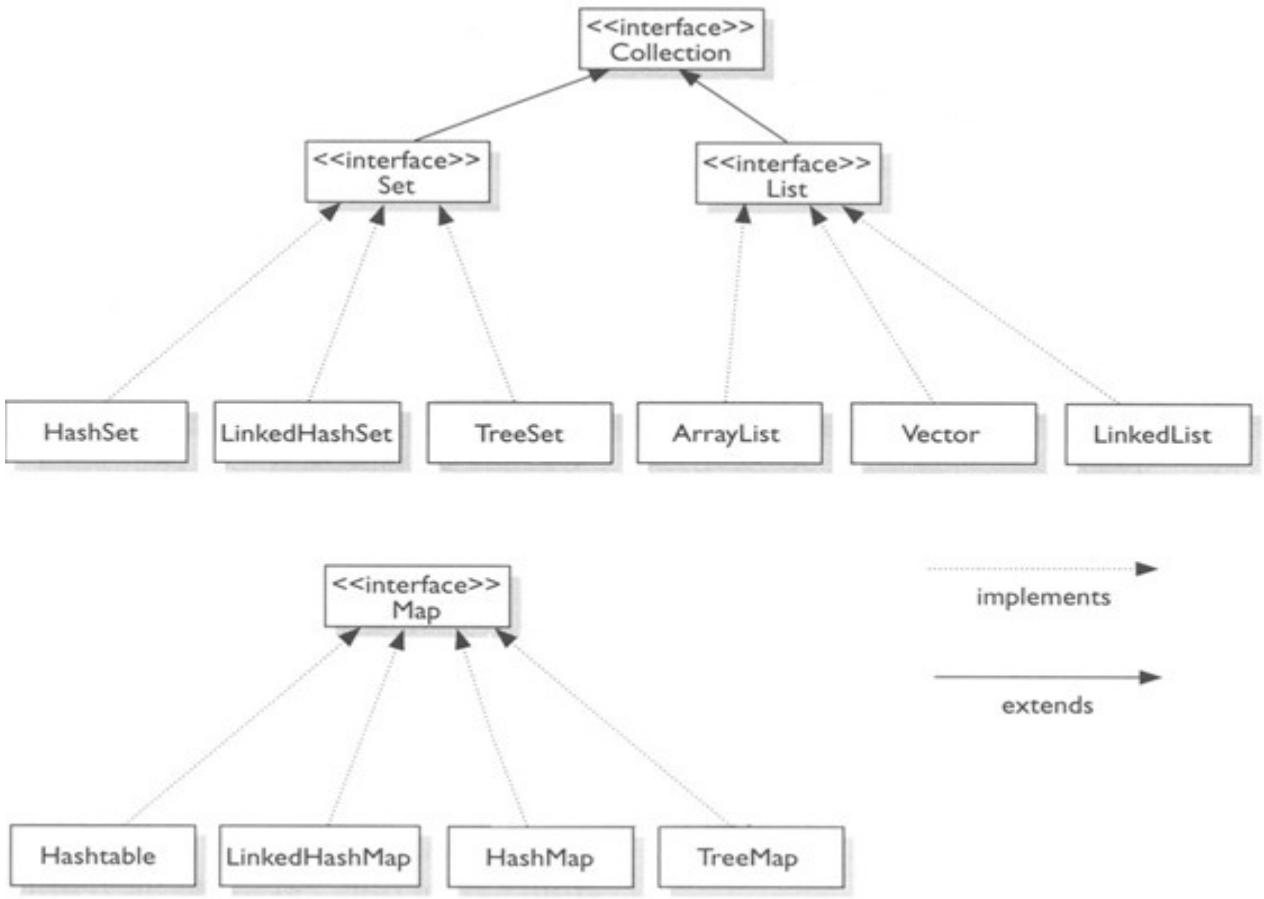
Listas – estructura secuencial, donde cada elemento tiene un índice o posición. Se utiliza el interfaz `List<E>`. Podemos utilizar las siguientes implementaciones:

- `ArrayList<E>` (acceso rápido)
- `LinkedList<E>`(inserciones/borrado rápidas)
- `Stack<E>` (pila), `Vector<E>` (obsoleto)

Diccionario o Matriz asociativa – cada elemento tiene asociado una clave que usaremos para recuperarlo. Se utiliza el interfaz `Map<K,V>`. Podemos utilizar las siguientes implementaciones:

`HashMap<K,V>`
`TreeMap<K,V>`
`LinkedHashMap<K,V>`

En el siguiente diagrama se muestra las relaciones de herencia entre los interfaces y clases más importantes en el framework de las colecciones.



Como puede verse en el esquema anterior los interfaces `List<E>` y `Set<E>` heredan del interface `Collection<E>`. A continuación se enumeran los métodos comunes a los interfaces `List<E>` y `Set<E>`, que son recogidos en el interface `Collection`. Supondremos que el elemento `e` es un objeto de la clase `E`:

`boolean` `add(E e)`: Añade un nuevo elemento al final de la lista.
`boolean` `remove(E e)`: Elimina la primera ocurrencia del elemento indicado.
`boolean` `contains(E e)`: Comprueba si el elemento especificado está en la colección.
`void` `clear()`: Elimina todos los elementos de la colección.

`int size()`: Devuelve el número de elementos en la colección.
`boolean isEmpty(Collection<?> c)`: Comprueba si la colección está vacía.
Los siguientes métodos combinan dos colecciones:
`boolean addAll(Collection<?> c)` : Añade todos los elementos de la colección `c`.
`boolean removeAll(Collection<?> c)`: Elimina todos los elementos de la colección `c`.
`boolean containsAll(Collection<?> c)`: Comprueba si coinciden las colecciones.
`boolean retainAll(Collection<?> c)`: Elimina todos los elementos a no ser que estén en `c`. (obtiene la intersección).

Conjuntos

Los conjuntos son estructuras de datos donde los elementos no tienen un orden y no se permiten duplicados. Para definirlos se utiliza la interfaz Set<E>, que no añade nuevos métodos a la interfaz `Collection<E>`. Por lo tanto, con los métodos anteriores podrás manipular tus conjuntos.

Podemos utilizar diferentes implementaciones. La más eficiente es `HashSet<E>` (implementación con tabla hash). Pero si queremos que los elementos queden ordenados podemos usar `TreeSet<E>` (implementación con árbol).
El siguiente ejemplo muestra cómo crear un conjunto de `Strings` y luego recorrerlo para mostrarlo en consola:

```
Set<String> conjunto = new HashSet();
 conjunto.add("manzana");

 conjunto.add("pera");

 conjunto.add("fresa");

 conjunto.add("naranja");

 conjunto.remove("pera");

 for(String s : conjunto) {

    System.out.println(s);

 }
```

Listas

Una lista es una estructura secuencial, donde cada elemento tiene un índice o posición. También recibe el nombre de array o vector unidimensional. El índice de una lista es siempre un entero y el primer elemento ocupa la posición 0.

Para trabajar con ellas se utiliza el interfaz `List<E>`. Las implementaciones más recomendables son: `ArrayList<E>`si queremos acceder a una posición de forma muy rápida o `LinkedList<E>` si queremos inserciones y borrado muy rápidos.

La interfaz `List<E>` hereda todos los métodos de `Collection<E>` y añade los siguientes:

> **`boolean add(int indice, E e):`** Inserta un nuevo elemento en una posición. El elemento que estaba en esta posición y los siguientes pararán a la siguiente.
> `E get(int indice):` Devuelve el elemento en la posición especificada.
> **`int indexOf(E e):`** Primera posición en la que se encuentra un elemento; -1 si no está.
> **`int lastIndexOf(E e):`** Última posición del el elemento especificado; o -1 si no está.
> `E remove(int indice):` Elimina el elemento de la posición indicada.
> `E set(int indice, E e):` Pone un nuevo elemento en la posición indicada. Devuelve el elemento que se encontraba en dicha posición anteriormente.

El siguiente ejemplo muestra como crear una lista de complejos y luego recorrerla:

```
List<Complejo> lista = new ArrayList<Complejo>();

lista.add( new Complejo(1.0, 5.0) );

lista.add( new Complejo(2.0, 4.2) );

lista.add(1, new Complejo(3.0, 0.0) );lista.remove(0);

for(Complejo c: lista) {

   System.out.println(c);

}
```

Observa como las dos primeras llamadas al método `add()` no se indica en qué posición de la lista se inserta. En estos casos se inserta al final de la lista. La tercera llamada se inserta en la posición 1, el elemento en esta posición y los siguientes son desplazados hacia atrás.

Como puedes ver la diferencia entre un conjunto y una lista es que en el conjunto los elementos no tienen una posición asignada; solo podemos ver si están o no están. Mientras que en la lista los elementos están ordenados según su posición. Por lo tanto, podemos insertar, consultar, eliminar o reemplazar elementos de una posición determinada.

Diccionarios o Mapas

Los diccionarios (también conocidos como mapas o matrices asociativas) son estructuras de datos donde cada elemento tiene asociado una clave que usaremos para recuperarlo (en lugar del índice de una lista). Para definirlos se utiliza la interfaz `Map<K,V>`. En este caso se trabaja con dos clases una que se utiliza como clave (`K`) y otra para almacenar los valores (`V`). La idea es que cada elemento se almacena mediante un par de objetos (`K,V`). Esta estructura de datos nos permite obtener el objeto `V` muy rápidamente, a partir de su clave `K`. Por ejemplo, podríamos almacenar objetos de la clase `Vehiculo` y utilizar como clave su matrícula en un `String`. De esta forma, a partir de la matrícula un diccionario encontraría el vehículo asociado muy rápidamente.

Podemos utilizar las siguientes implementaciones de este interfaz:

```
HashMap<K,V>
TreeMap<K,V>
LinkedHashMap<K,V>
```

Veamos los métodos del interfaz `Map<K,V>` (Ojo. A diferencia de otras colecciones no hereda del interfaz `Collection<E>`)).

`V put(K key, V value)`: Añade un nuevo par clave-valor al diccionario.
`V get(Object key)`: Da el valor asociado a una clave o **`null`** si no se encontró.
`V remove(Object key)`: Elimina el par clave-valor que corresponde a la clave.
`boolean` `containsKey(Object key)`: Comprueba si está la clave especificada.
`boolean` `containsValue(Object value)`: Comprueba si está el valor.

`Set keySet()`: Devuelve un conjunto con las claves contenidas en el diccionario.
`Collection values()`: Devuelve una colección con solo los valores.
`boolean` `isEmpty()`: Comprueba si la colección está vacía.
`int` `size()`: Devuelve el número de elementos que contiene la colección.
`void` `clear()`: Elimina todos los elementos de la colección.

El siguiente ejemplo muestro como crear un diccionario para almacenar objetos de la clase `Vehiculo` utilizando como clave un `String` con la matrícula:

```
Map diccionario = new HashMap();

diccionario.put("V 1245", new Vehiculo());

diccionario.put("A 2455", new Vehiculo());

diccionario.get("V 1245");
```

La clase estática `Collections` nos ofrece herramientas para ordenar y buscar en colecciones. Los interfaces `Iterator` y `ListIterator`facilitan recorres colecciones para hacer bucles.