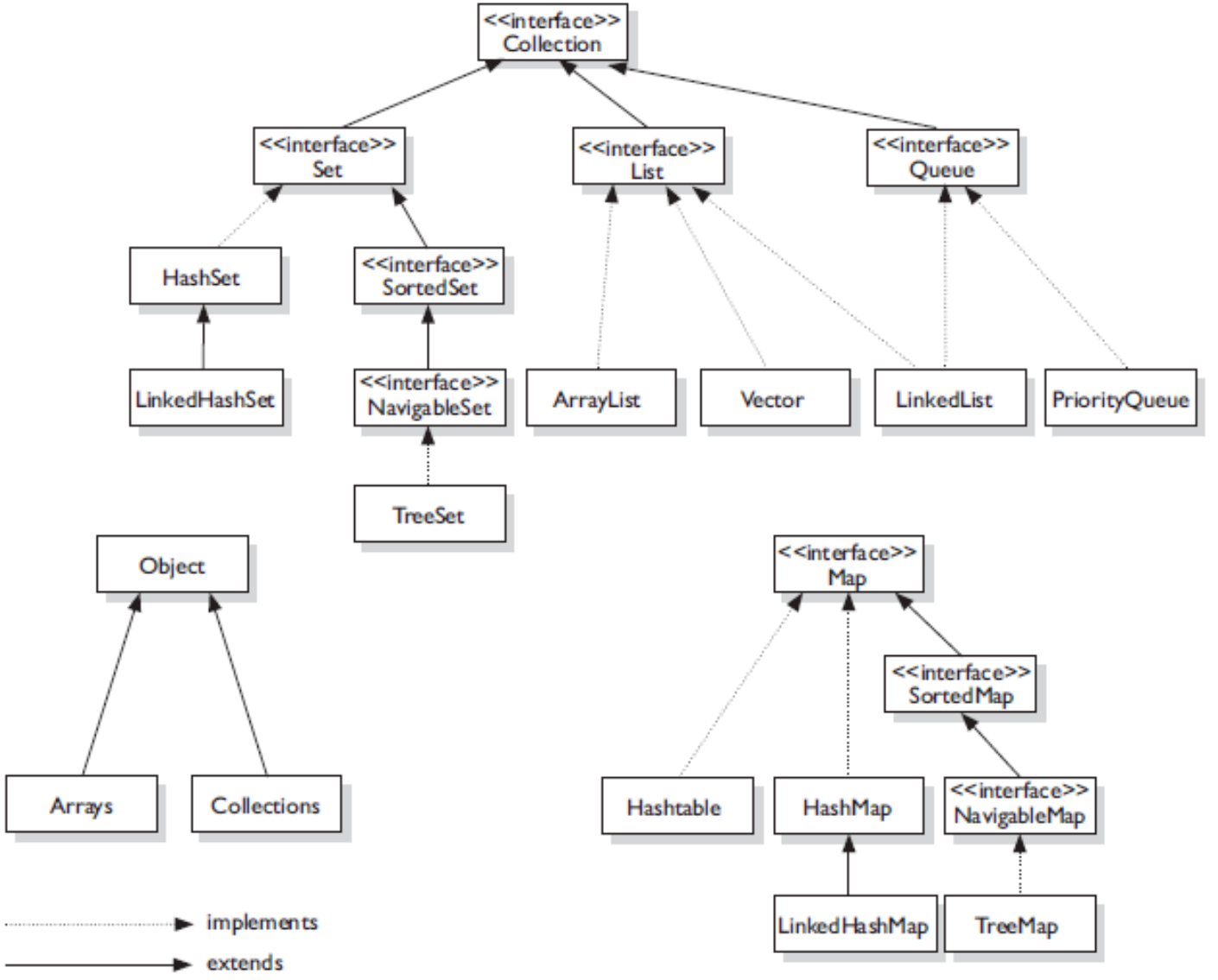
Collections

El API Collections fue introducido con la versión 1.2 de Java 2 Standar Edition y su objetivo fundamental es proporcionar una serie de clases e interfaces que nos permitan manejar diversos conjuntos de datos de una manera estandar. Dentro de esta API nos vamos a encontrar tanto con estructuras de datos como con algoritmos para utilizar dichas estructuras.

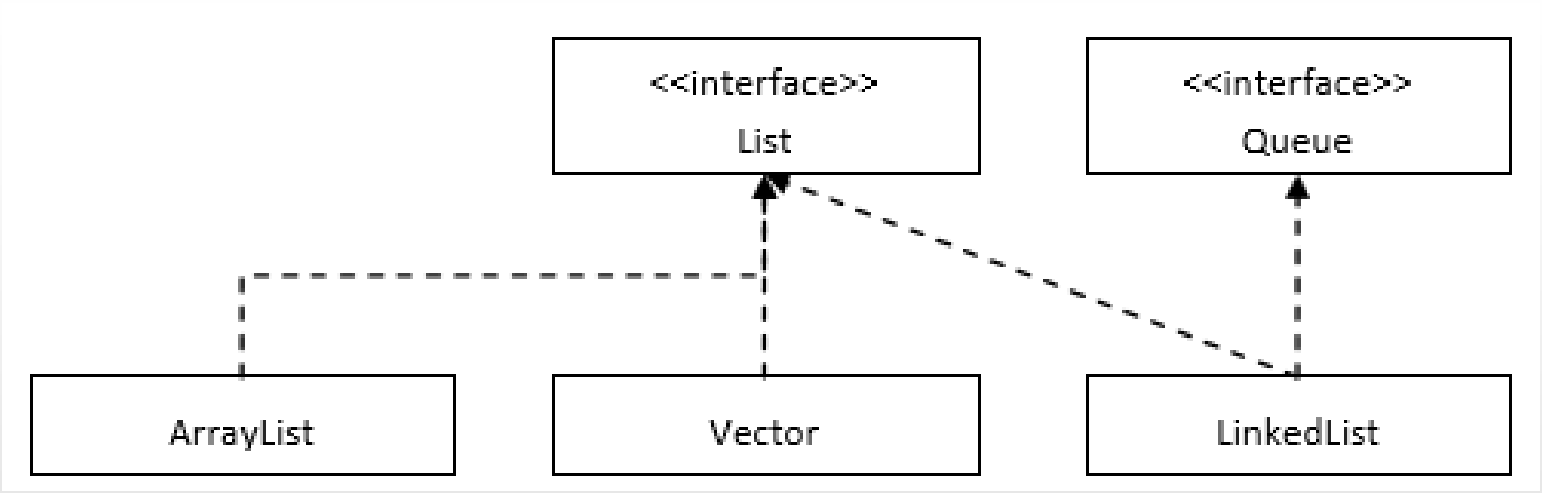
La base del API Collections está formada por un conjunto de interfaces para trabajar con conjuntos de datos. Entender el funcionamiento de estas interfaces significará entender el funcionamiento de la totalidad de la API. En la siguiente figura podemos ver la estructura de estas interfaces.

Collection Framework



List

Todas las listas manejan sus colecciones mediante índices. Permiten agregar un elemento dentro de la colección en un índice específico, obtener un elemento de la colección según el índice, y buscar en que posición se encuentra un objeto en particular.



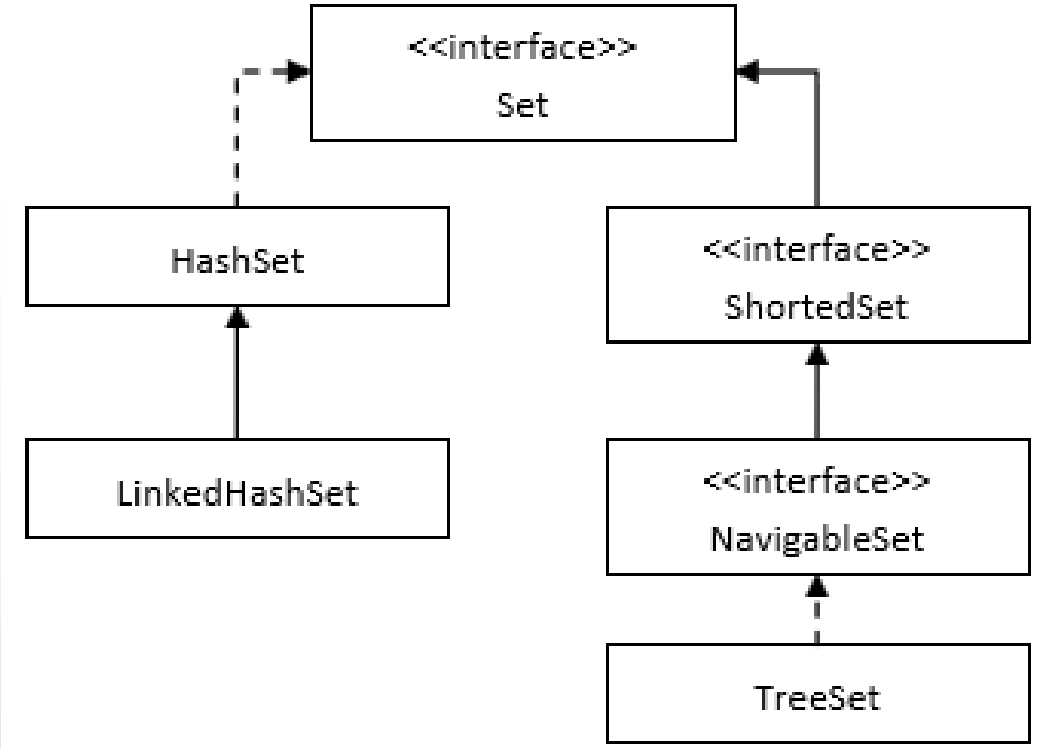
ArrayList: Funciona como un array que puede modificar su tamaño dinámicamente. Permite acceder aleatoriamente a sus elementos.

Vector: Se comporta exactamente igual que el ArrayList, con la salvedad de que es Thread-Safe (multihilo seguro).

LinkedList: Mantiene un índice aparte, de manera que se puede utilizar como una cola, o como un simple ArrayList. Añade métodos como agregar al comienzo o al final.

Set

Los sets solo manejan objetos únicos, impidiendo su duplicidad dentro de la colección. Para saber si un objeto ya se encuentra agregado en la colección, es necesario que se implemente el método equals().



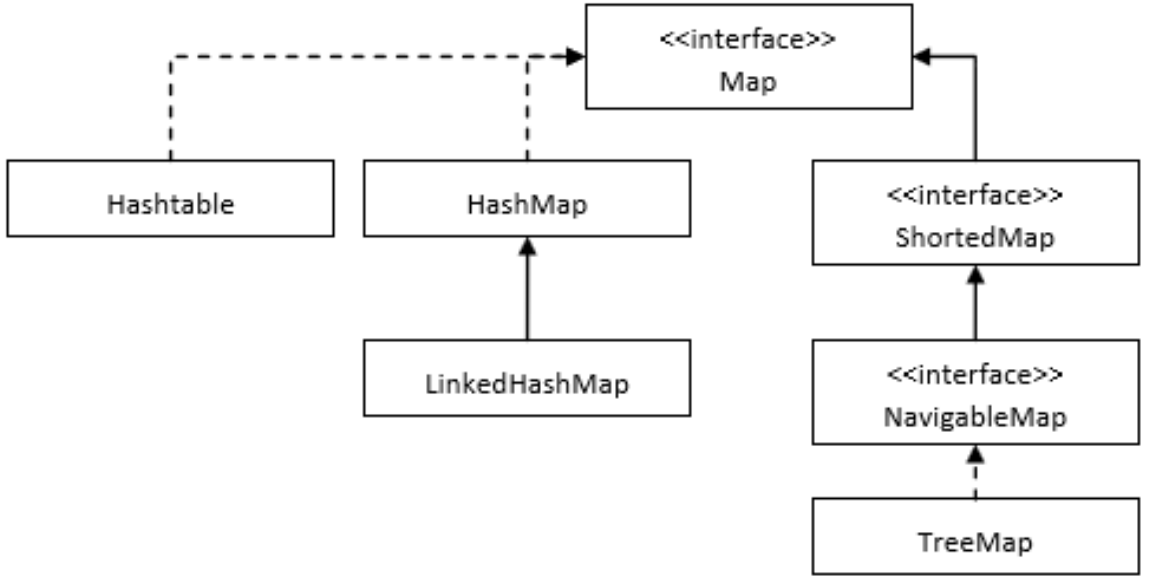
HashSet: Los códigos se almacenan sin un orden definido, utilizando como identificador su código de hash.

LinkedHashSet: Se comporta como HashSet con la salvedad de que esta mantiene otro índice, el cual almacena el orden de los elementos según fueron siendo insertados en la colección.

TreeSet: Mantiene los elementos ordenados según su “orden natural”. Los objetos almacenados en esta colección requieren implementar Comparable o Comparator.

Map

Los maps corresponden con la funcionalidad de los sets, con la diferencia de que los maps no tienen restricciones en cuanto a la duplicidad de sus elementos. Al igual que los sets, requiere que se implemente equals() y hashCode().



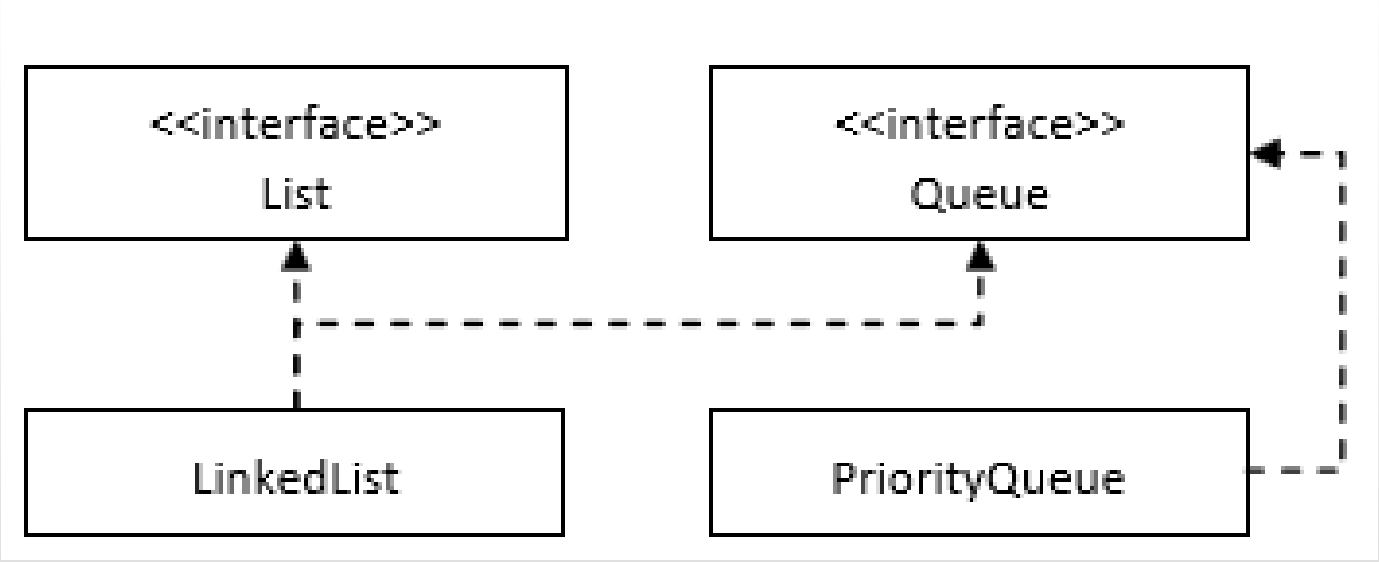
HashMap: Los códigos se almacenan sin un orden definido, utilizando como identificador su código de hash.

Hashtable: Se comporta como HashMap, con la salvedad de que es Thread-Safe (multihilo seguro).

LinkedHashMap: Se comporta como HashMap con la salvedad de que esta mantiene otro índice, el cual almacena el orden de los elementos según fueron siendo insertados en la colección.
TreeMap Mantiene los elementos ordenados según su “orden natural”. Los objetos almacenados en esta colección requieren implementar Comparable o Comparator.

Queue

Representan un listado de “cosas por hacer”. El comportamiento puede variar, pero se comportan por defecto como una cola o FIFO, First-In – First-Out (Primero en entrar – primero en salir).



PriorityQueue: Es una lista por defecto FIFO con prioridad. Requiere la implementación de Comparable.