Colecciones en Java

Colecciones en Java

- Se utilizan para agrupar objetos en una sola unidad • Representan

datos que forman un grupo natural ● Similares a los arrays, pero de tamaño dinámico ● La jerarquía de colecciones en Java incluye:

- Interfaces
- Implementaciones
- Algoritmos

Colecciones en Java - Interfaces

● Las interfaces Collection son tipos de datos abstractos que representan las colecciones. ● Permiten manipular las colecciones independientemente de los detalles de su implementación (comportamiento polimórfico) ● Java define un conjunto de interfaces:

- Collection

- Set
- List
- Map

Colecciones en Java - Implementaciones

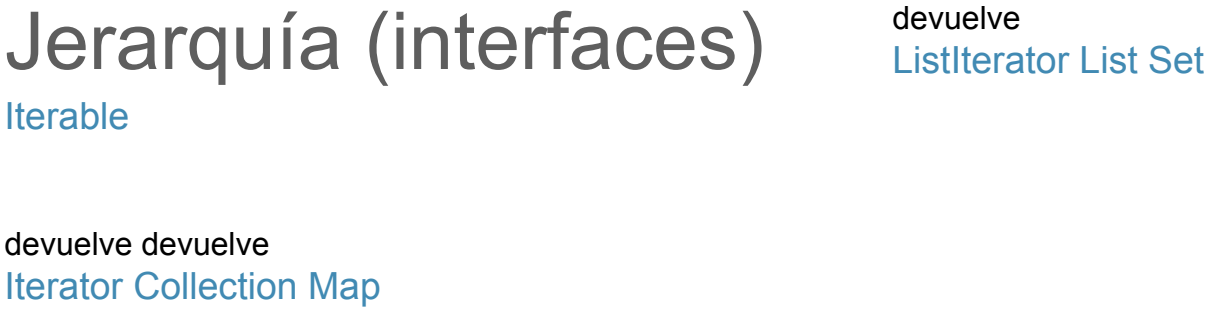
- Existen implementaciones concretas de las interfaces, que son estructuras de datos reusables
  - ArrayList
  - Vector

Las que vienen usando hasta ahora

Colecciones en Java - Algoritmos

- Son métodos polimórficos que realizan computaciones útiles en objetos que implementan las interfaces collection
- Los algoritmos más comunes son:
  - Ordenar (sorting)
  - Mezclar (shuffling)
  - Manipulaciones de rutina (invertir, rellenar, copiar, intercambiar, añadir)
  - Búsqueda (searching)
  - Valores extremos (min, max)

Colecciones en Java – Jerarquía (interfaces)



SortedMap

SortedSet

Colecciones en Java – Jerarquía (abstracciones)

Iterable

devuelve devuelve

Iterator Collection Map devuelve

ListIterator List Set

SortedMap

SortedSet

Colecciones en Java – Jerarquía (implementaciones)

Iterable

devuelve devuelve

Iterator Collection Map devuelve

ListIterator List Set

SortedSet

SortedMap

Colecciones en Java – Clases de utilidades

- Ordenamiento
  - Collections.sort(List l)
  - Collections.sort(List l, Comparator c)

Mezclar

- Collections.shuffle(List l)

- Manipulaciones de rutina

Collections.reverse(List l)

- Collections.fill(List l, Object o)

Compartor Comparable

- Collections.copy(List destino, List origen)

Collextions.swap(List l, int i, int j)

Colecciones en Java – Clases de utilidades

Compartor Comparable

- Búsqueda
  - Collections.binarySearh(List l, Object o) → Asume que la lista está ordenada

ascendentemente por su orden natural

- Collections.binarySearch(List l, Object o, Comparator c) → Asume que la lista está ordenada ascendentemente de acuerdo al comparador ○ Retorna el índice del elemento encontrado

- Valores extremos
  - Collections.min(List l) → Asume que los elementos de la lista son Comparables
  - Collections.min(List l, Comparator c)
  - Collections.max(List l) → Asume que los elementos de la lista son comparables
  - Collections.max(List l, Comparator c)

Colecciones en Java – Clases de utilidades

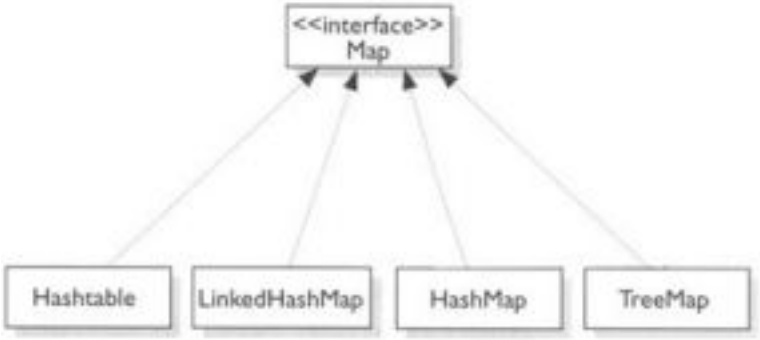
Compartor Comparable

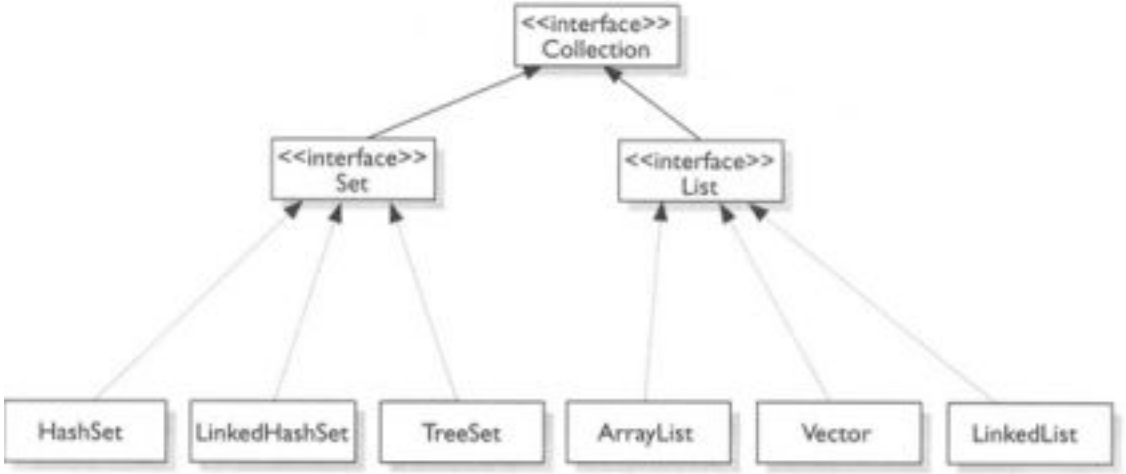
- Frecuencia
  - Collections.frequency(Collection c, Object o)

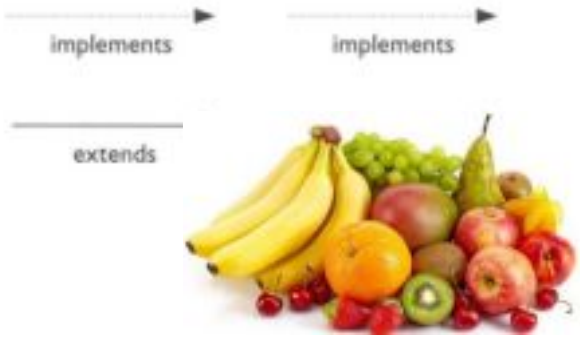
- Colecciones disjuntas

- Collections.disjoint(Collection c1, Collection c2)

Colecciones en Java – Jerarquía (simplificada)







Colecciones en Java

- **Collection<T>**: Un grupo de elementos individuales, frecuentemente con alguna regla aplicada a ellos. Las operaciones básicas de una Collection son:
  - **add(T t)**: Añade un elemento.
  - **addAll(Collection c)**: añade todos los elementos de c a la colección
  - **size()**: Obtiene la cantidad de elementos de la colección.

- **contains(T t)**: consulta si el elemento t ya está dentro de la colección. ○ **remove(T t)**: elimina el objeto t (primera ocurrencia)
- **clear():** elimina todos los elementos de la colección
- **isEmpty()**: consulta si la colección contiene elementos.
- **removeAll(Collection c)**: elimina todos los elementos que existen en c ○ **retainAll(Collection c)**: solo retiene los elementos que existen en c ○ **iterator()**: Obtiene un “iterador” que permite recorrer la colección visitando cada elemento una vez.

Colecciones en Java – Recorrido de colecciones

- for-each
  - Sintaxis similar al for, sin índices

- Iterator
    - Objetos que nos permiter recorrer una colección

Colecciones en Java

0 1 2 3

4 5

- **List<T>**: Elementos en una secuencia particular que mantienen un orden y permite duplicados. Añade métodos que permiten el acceso a los elementos por su posición y la búsqueda de elementos:
    - **get(int i)**: obtiene el elemento en la posición i.
    - **set(int i, T t)**: Pone al elemento t en la posición i (reemplaza). ○ **add(int i, T t)**: Inserta al elemento en la posición i (desplaza) ○ **add(T t)**: Agrega el elemento t al final de la lista

- **indexOf(T t)** busca un elemento concreto de la lista y devuelve su posición.
- **remove(int i)**: elimina el elemento de la posición i

Colecciones en Java

- Algunas implementaciones de List son:
  - **ArrayList<E>**
  - **Vector<E>**
  - **LinkedList<E>**

Colecciones en Java

- **Set<E>**

- define una colección que no puede contener elementos duplicados. ○ Si se intenta agregar un duplicado, no se agrega.
- No agrega métodos nuevos, pero si la restricción de que los elementos duplicados están prohibidos.
- Es necesario que los elementos tengan implementada, de forma correcta, los métodos equals y hashCode.
- Para comprobar si dos Set son iguales, se comprobarán si todos los elementos que componen son iguales sin importar en el orden que ocupen dic

Colecciones en Java

- Algunas implementaciones de Set son:

- **HashSet<E>**: Es la implementación con mejor rendimiento de todas, pero no garantiza ningún orden a la hora de realizar iteraciones. Es la implementación más empleada debido a su rendimiento y a que, generalmente, no nos importa el orden que ocupen los elementos.

- **TreeSet<E>**: esta implementación almacena los elementos ordenándolos en función de sus valores. Es bastante más lento que HashSet. Los elementos almacenados deben implementar la interfaz Comparable.

- **LinkedHashSet<E>**: esta implementación almacena los elementos en función del orden de inserción. Es un poco más costosa que HashSet.

Map<K,V>:

Colecciones en Java

- “Manzana” “Naranja” “Banana” “Kiwi”

- Un grupo de pares objeto clave-valor, que no

permite duplicados en sus claves.

- No utiliza la interfaz Collection.
- Los principales métodos son: **put(clave, valor)**, **get(clave)**, **remove(clave)**.
- Además hay funciones útiles para recorrer los elementos del map: ■ **keyset()** retorna un Set con todas las claves ■ **values()** retorna una Collection con todos los valores ■ **entrySet()** retorna un Set de objetos **Map.Entry**

Colecciones en Java

- Algunas implementaciones de Map son:
  - **HashMap<K,V>**: permite valores null.
  - **HashTable<K,V>**: no permite null.

“Manzana” “Naranja” “Banana” “Kiwi”

- **LinkedHashMap<K,V>**: Extiende de HashMap y utiliza una lista doblemente enlazada para recorrerla en el orden en que se añadieron. Es ligeramente más rápida a la hora de acceder a los elementos que su superclase, pero más lenta a la hora de añadirlos.
- **TreeMap<K,V>**: Se basa en una implementación de árboles en el que ordena los valores según las claves. Es la clase más lenta.

Colecciones en Java

- ¿Qué colección

uso?

Colecciones en Java

Pares

Valores

Si

No

Si

No

No

Si

No

Inserción

De clave

Si

De elementos

De inserción