

Colecciones





Listas





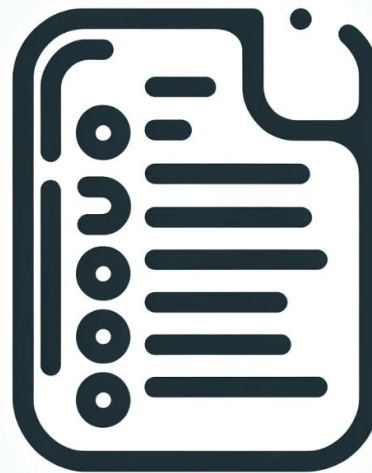
Listas

Definición: Colección de datos **ordenada**.

Propiedades interesantes:

- Mantiene el **orden** de inserción.
- **Acceso directo** a elementos por el índice.
- Permite la inserción de valores **duplicados y nulos**.
- **Iterable**.

ArrayList, Vector, Stack, LinkedList



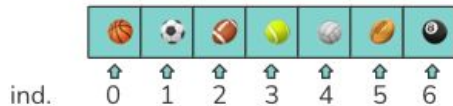


Clase: ArrayList

Definición: Array de tamaño variable

Propiedades interesantes:

- Array de tamaño “dinámico” 🎉
- Rápidos en la obtención y modificación de datos
- Lento en la inserción o eliminación de datos





ArrayList: Métodos

- **Añadir elementos**
 - void add(int posición, E elemento)
 - boolean add(E elemento)
 - boolean addAll(int posición, Collection<? extends E> c2)
- **Recuperar elemento en una posición**
 - E get(int posición)
- **Devuelve el (primer o último) índice de un elemento**
 - int indexOf(Object o)
 - int lastIndexOf(Object o)
- **Reemplaza elementos en una posición**
 - E set(int posición, E elemento)
- **Ordena los elementos de un array**
 - sort(Comparator<? super E> c)
- **Elimina los elementos situados en una posición**
 - E remove(int posición)

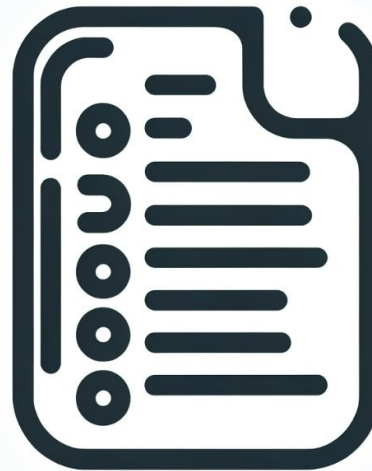
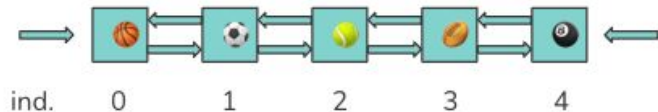


LinkedList

Definición: Implementación de lista con nodos de doble enlace.

Propiedades interesantes:

- ~~Array~~ de tamaño dinámico 🎉🎉
- Rápidos en la inserción o eliminación de datos cabeza/cola
- Lentos en el acceso directo / modificación intermedios
- Puede actuar como una Cola/Pila





LinkedList: Métodos

- **Inserta elementos**
 - `addFirst(E e)`
 - `addLast(E e)`
- **Devuelve elementos**
 - `E getFirst()`
 - `E getLast()`
- **Elimina elementos**
 - `E removeFirst()`
 - `E removeLast()`
- **Recorre descendentemente**
 - `Iterator<E> descendingIterator()`
- **Elimina la primera/última ocurrencia**
 - `boolean removeFirstOccurrence(Object o)`
 - `boolean removeLastOccurrence(Object o)`

Sets



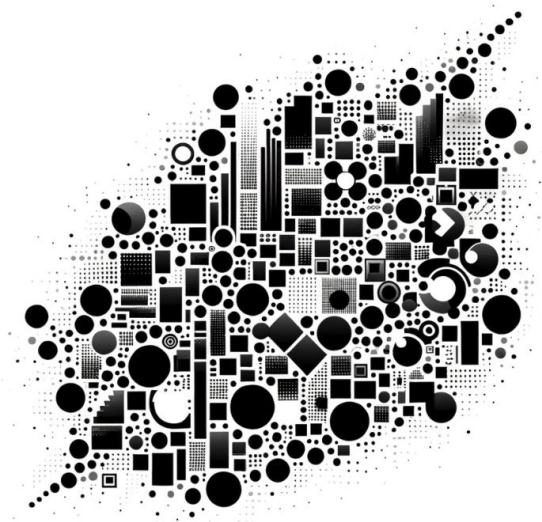
Sets

Definición: Colección de datos **únicos**.

Propiedades interesantes:

- Mantiene la **unicidad**.
- **No ordenado** *.
- Operaciones de conjuntos eficientes.
- **Iterable**.

HashSet, LinkedHashSet, TreeSet



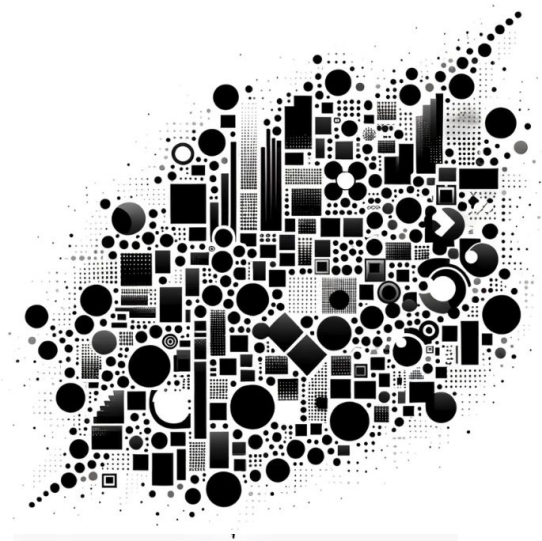
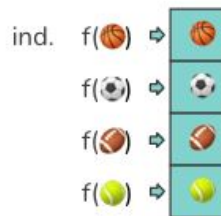


Clase: HashSet

Definición: Utiliza una tabla Hash para almacenar elementos

Propiedades interesantes:

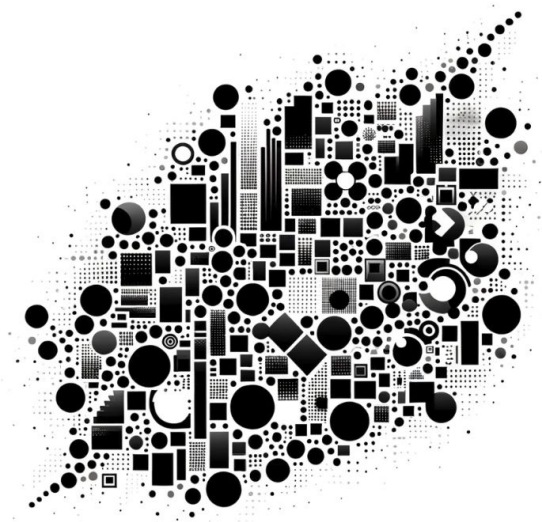
- Rápidos añadiendo, borrando y consultando.





HashSet: Métodos

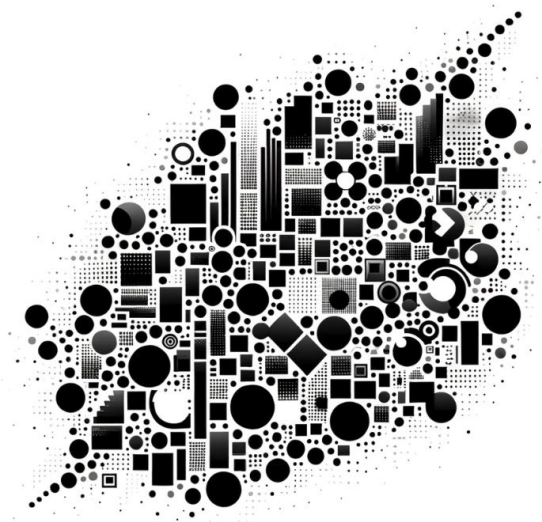
- **Inserta elementos**
 - `add(E e)`
- **Consulta existencia**
 - `boolean contains(E e)`
 - `boolean isEmpty()`
- **Elimina elementos**
 - `boolean remove(E e)`





HashSet: Usos

- Eliminar duplicados
- Comprobar existencia en conjunto grande de datos
- Operaciones sobre conjuntos: unión, intersección...



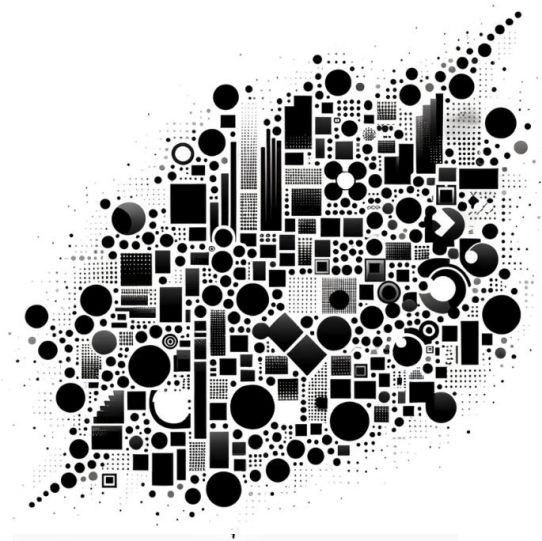
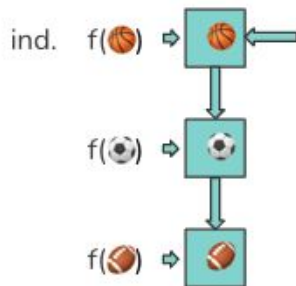


Clase: LinkedHashSet

Definición: Utiliza una tabla Hash enlazada

Propiedades interesantes:


- Pérdida de rendimiento añadiendo, borrando y consultando.
- Garantiza el **orden** 🎉 🎉





LinkedHashSet: Métodos

- **Inserta elementos**
 - add(E e)
- **Consulta existencia**
 - boolean contains(E e)
 - boolean isEmpty()
- **Elimina elementos**
 - boolean remove(E e)

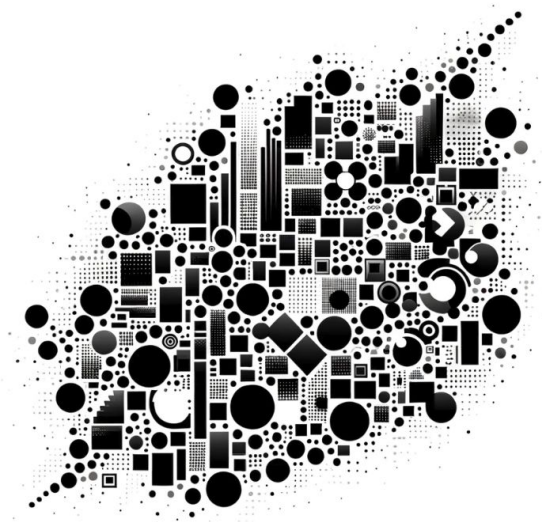


```
public class LinkedHashSet<E>  
extends HashSet<E>
```



LinkedHashSet: Usos

- Eliminar duplicados y orden de inserción
- Cache LRU: Eliminar datos más antiguos
- Mantener histórico de eventos



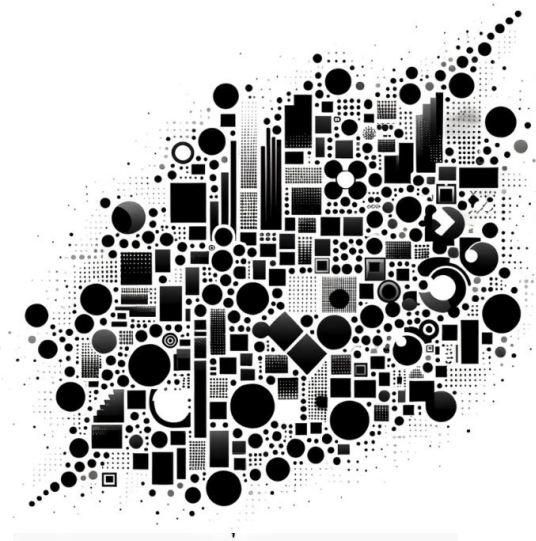
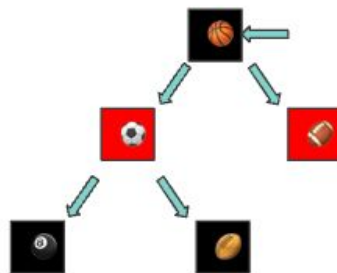


Clase: TreeSet

Definición: Utiliza un árbol rojo-negro para almacenar elementos

Propiedades interesantes:

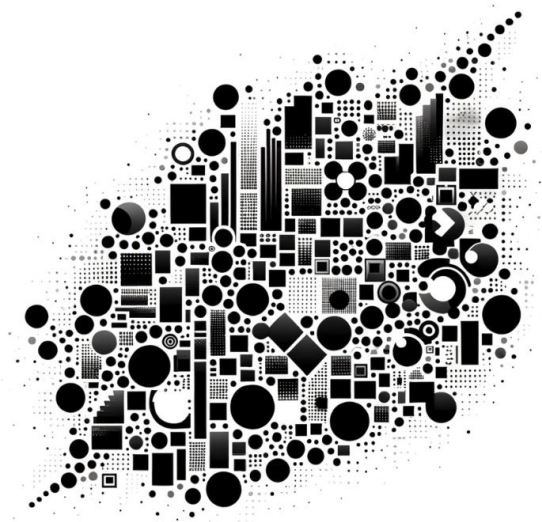
- El uso del árbol binario mejora el rendimiento búsqueda.
- Garantiza el **orden** 🎉 🎉
- No puede contener **null**





TreeSet: Métodos

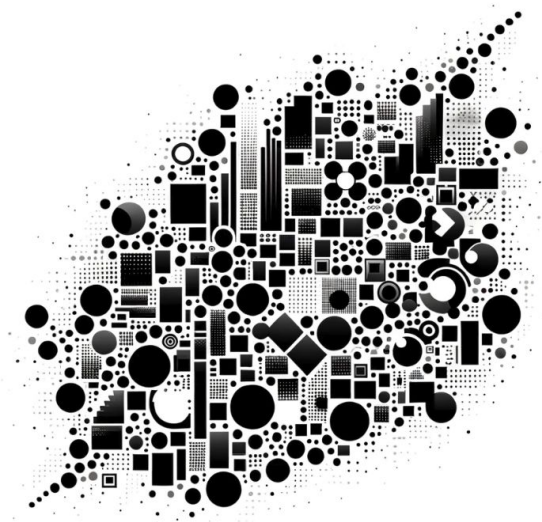
- **Inserta elementos**
 - `add(E e)`
- **Elimina elementos**
 - `remove(E e)`
- **Obtener elementos**
 - `first(E e)`
 - `last(E e)`
 - `subSet(E from, E to)`
- **Consulta existencia**
 - `boolean contains(E e)`
 - `boolean isEmpty()`
- **Elimina elementos**
 - `boolean remove(E e)`





TreeSet: Usos

- Eliminar duplicados y orden customizable
- Búsqueda eficiente de elementos y rangos



Mapas





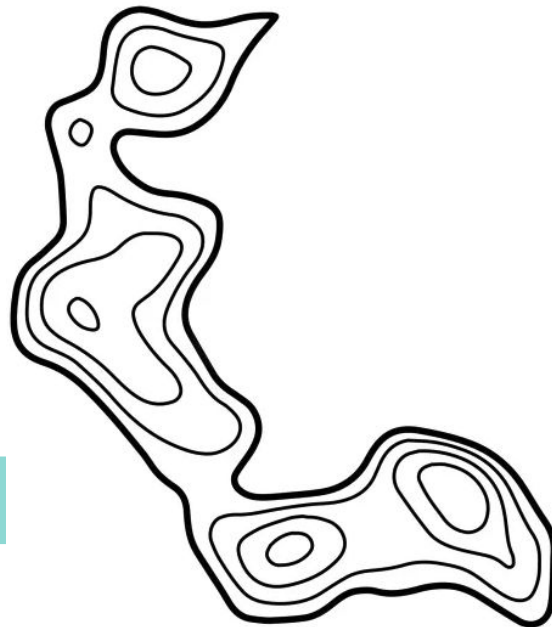
Mapas

Definición: Almacenamiento de pares clave-valor de manera que es fácil obtener un valor a través de la clave.

Propiedades interesantes:

- La clave-valor es 1:1 (una clave sólo puede tener un valor).
- **Acceso eficiente** a los valores.
- **Iterable**.

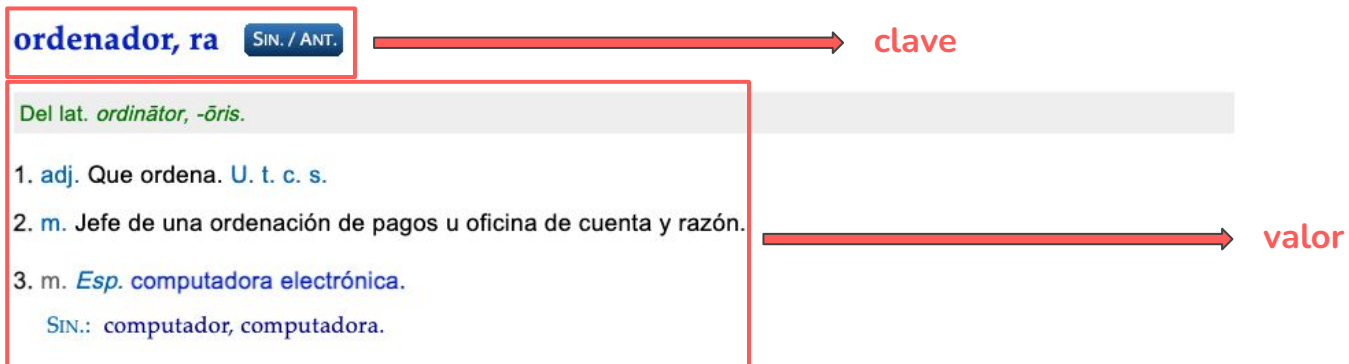
HashMap, LinkedHashMap, TreeMap





Mapas

En el ámbito de la programación, los Mapas también se conocen como **Diccionarios** por la similitud funcional.





Clase: HashMap

Definición: Utiliza una tabla Hash para almacenar la clave-valor.

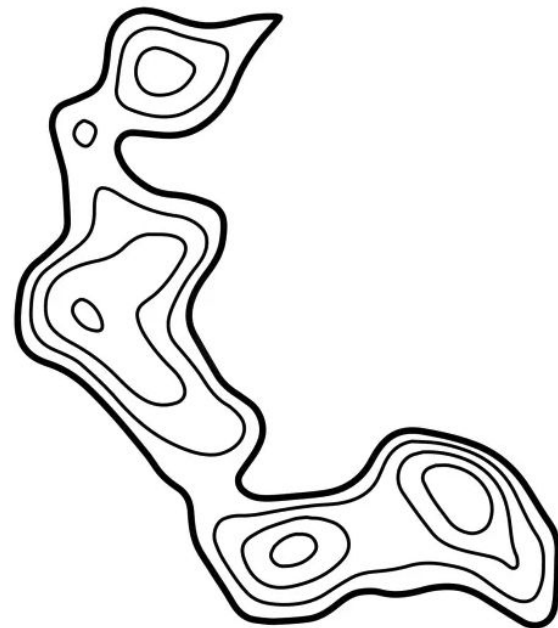
Propiedades interesantes:

- Rápidos añadiendo, borrando y consultando.

<🏀, 🏃>

<⚽, 🏃>

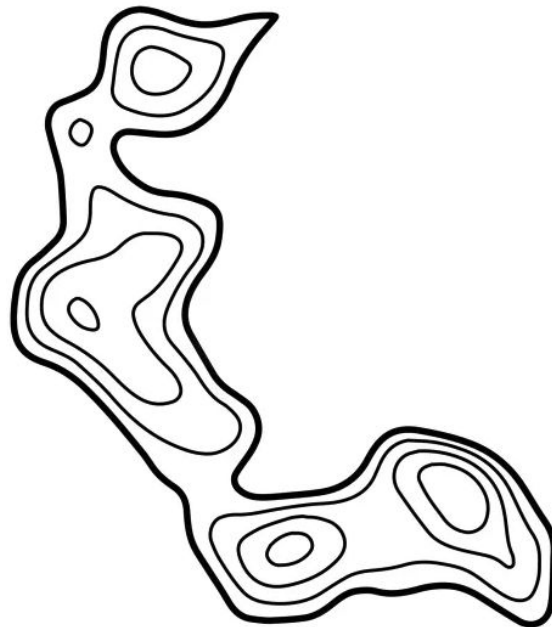
<🏏, 🏃>





HashMap: Métodos

- **Inserta elementos**
 - K put(K key, V value)
 - V replace(K key, V value)
- **Consulta elementos**
 - E get(O key)
 - boolean isEmpty()
 - int size()
 - Collection<V> values()
 - boolean containsKey(K key)
- **Elimina elementos**
 - O remove(O key)





Clase: HashMap

Definición: Utiliza una tabla Hash para almacenar la tupla

Propiedades interesantes:

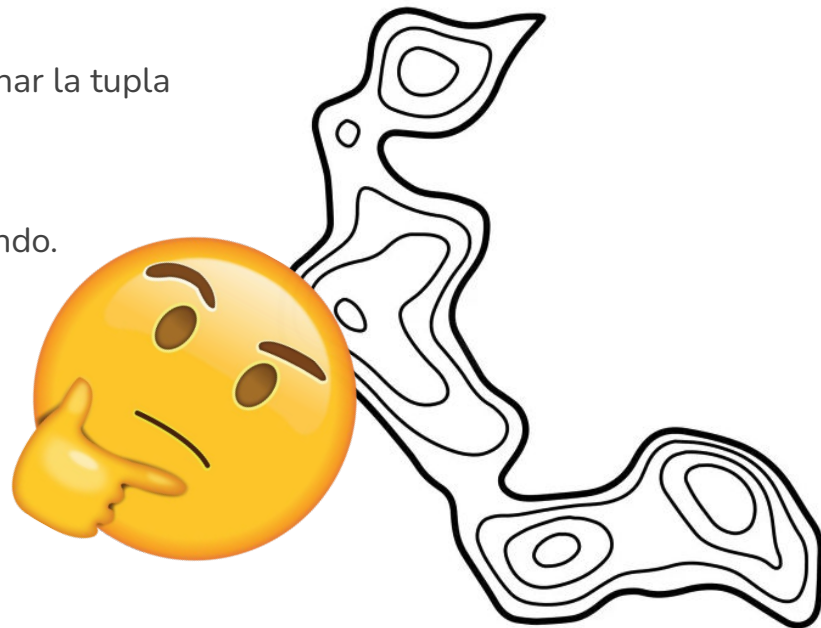
- Rápidos añadiendo, borrando y consultando.

<🏀, 🏃>

<⚽, 🏃>

<🏏, 🏃>


ind.





HashMap

HashSet usa internamente un HashMap

hashSet.add(🏀)  hashMap.put(🏀, "")

<🏀, ">

Para el HashSet el valor en el mapa es irrelevante porque la clave ya tiene la información del elemento del conjunto.

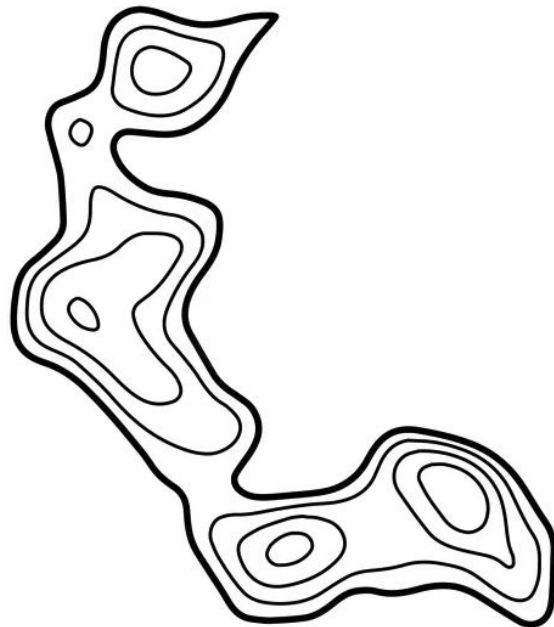


Clase: LinkedHashMap

Definición: Utiliza una tabla Hash enlazada para almacenar las clave-valor.

Propiedades interesantes:

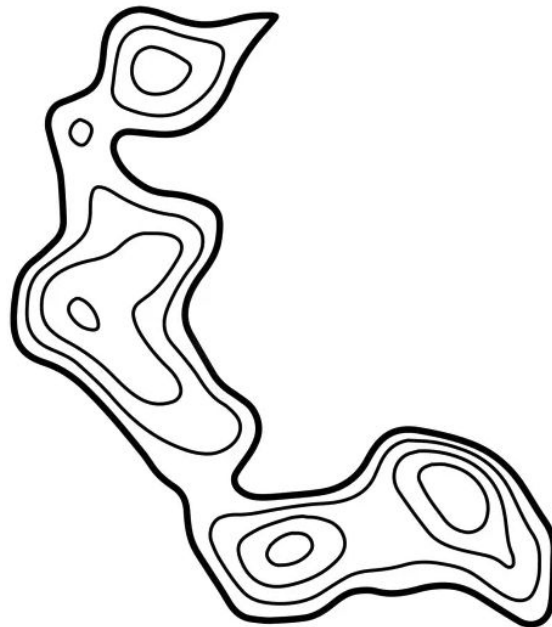
- Mantiene el orden de inserción.





LinkedHashMap: Métodos

- **Inserta elementos**
 - K put(K key, V value)
 - V replace(K key, V value)
- **Consulta elementos**
 - E get(O key)
 - boolean isEmpty()
 - int size()
 - Collection<V> values()
 - boolean containsKey(K key)
- **Elimina elementos**
 - O remove(O key)





Clase: LinkedHashMap

Definición: Utiliza una tabla Hash enlazada para almacenar las clave-valor.

Propiedades interesantes:

- Mantiene el orden de inserción.

<🏀, 🏃>


<⚽, 🏃>

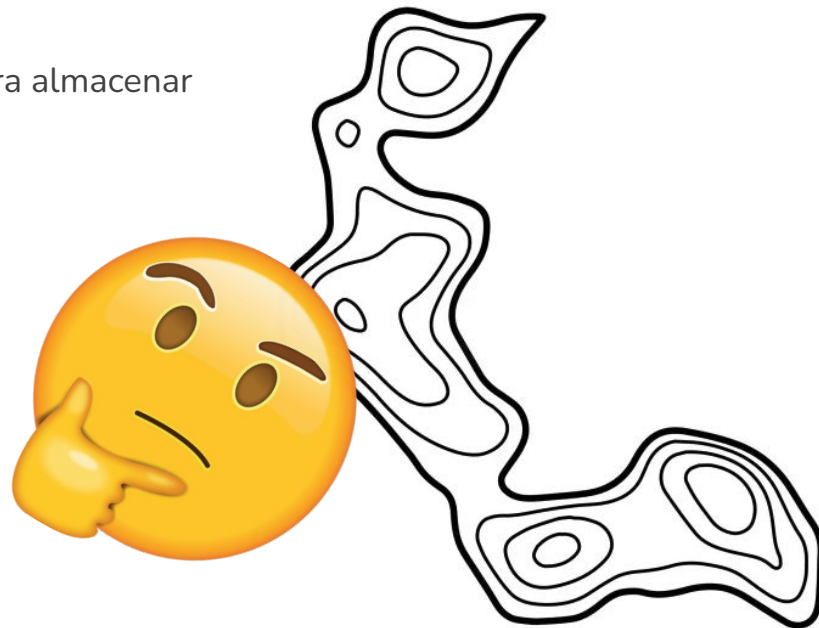
<🎱, 🏃>

ind.

f(🏀) → 

f(⚽) → 

f(🎱) → 





LinkedHashMap

LinkedHashSet usa internamente un LinkedHashMap

`linkedHashSet.add(🏀)`  `linkedHashMap.put(🏀, "")`
`<🏀, ">`

Para el LinkedHashSet el valor en el mapa es irrelevante porque la clave ya tiene la información del elemento del conjunto.

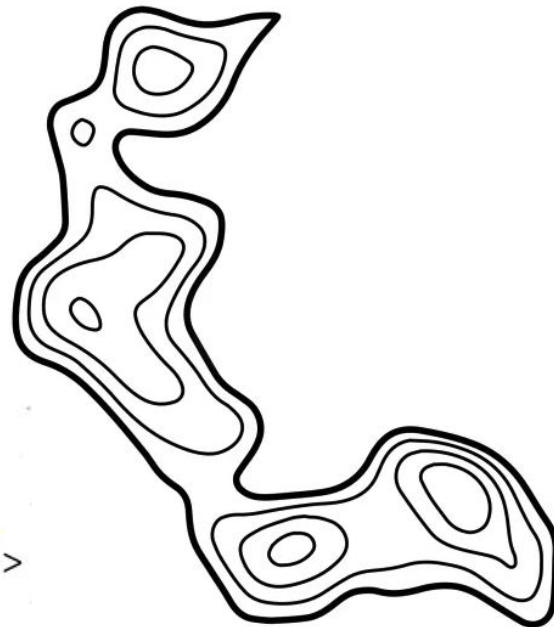


Clase: TreeMap

Definición: Utiliza un árbol rojo-negro para almacenar los los elementos.

Propiedades interesantes:

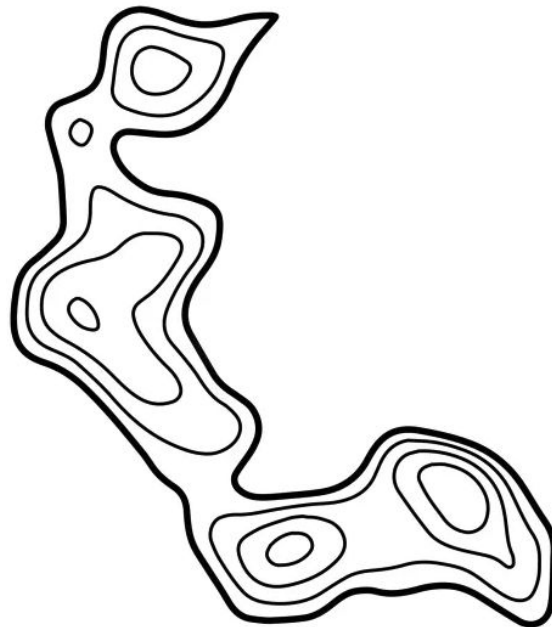
- El uso del árbol binario mejora el rendimiento búsqueda.
- Garantiza el **orden** 🎉 🎉
- No puede contener **null**





TreeMap: Métodos

- **Inserta elementos**
 - K put(K key, V value)
 - V replace(K key, V value)
- **Consulta elementos**
 - E get(O key)
 - boolean isEmpty()
 - int size()
 - Collection<V> values()
 - boolean containsKey(K key)
 - K firstKey()
 - K lastKey()
- **Elimina elementos**
 - O remove(O key)





Clase: TreeMap

Definición: Utiliza un árbol rojo-negro para almacenar las clave-valor.

Propiedades interesantes:


- El uso del árbol binario mejora el rendimiento búsqueda
- Garantiza el **orden** 🎉 🎉
- No puede contener **null**





TreeMap

TreeSet usa internamente un TreeMap

`treeSet.add(🏀)`  `treeMap.put(🏀, "")`
`<🏀, ">`

Para el TreeSet el valor en el mapa es irrelevante porque la clave ya tiene la información del elemento del conjunto.