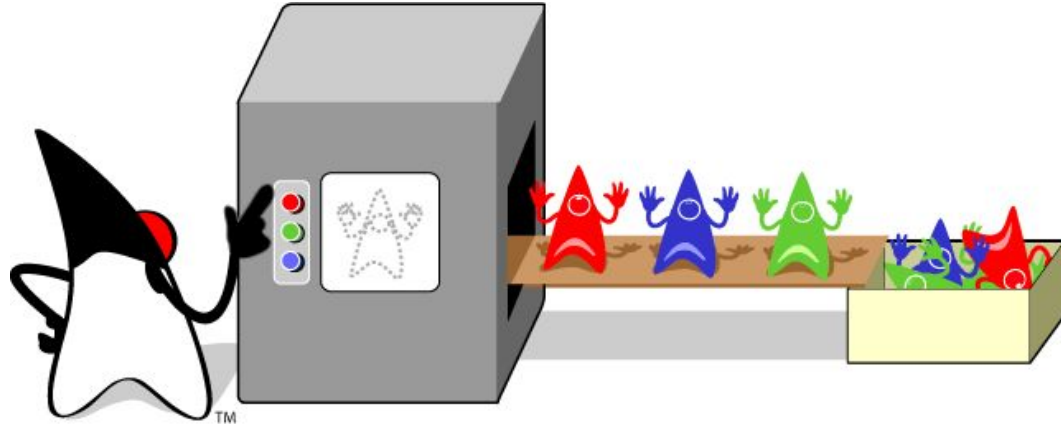

Clase 14: Collection - Set

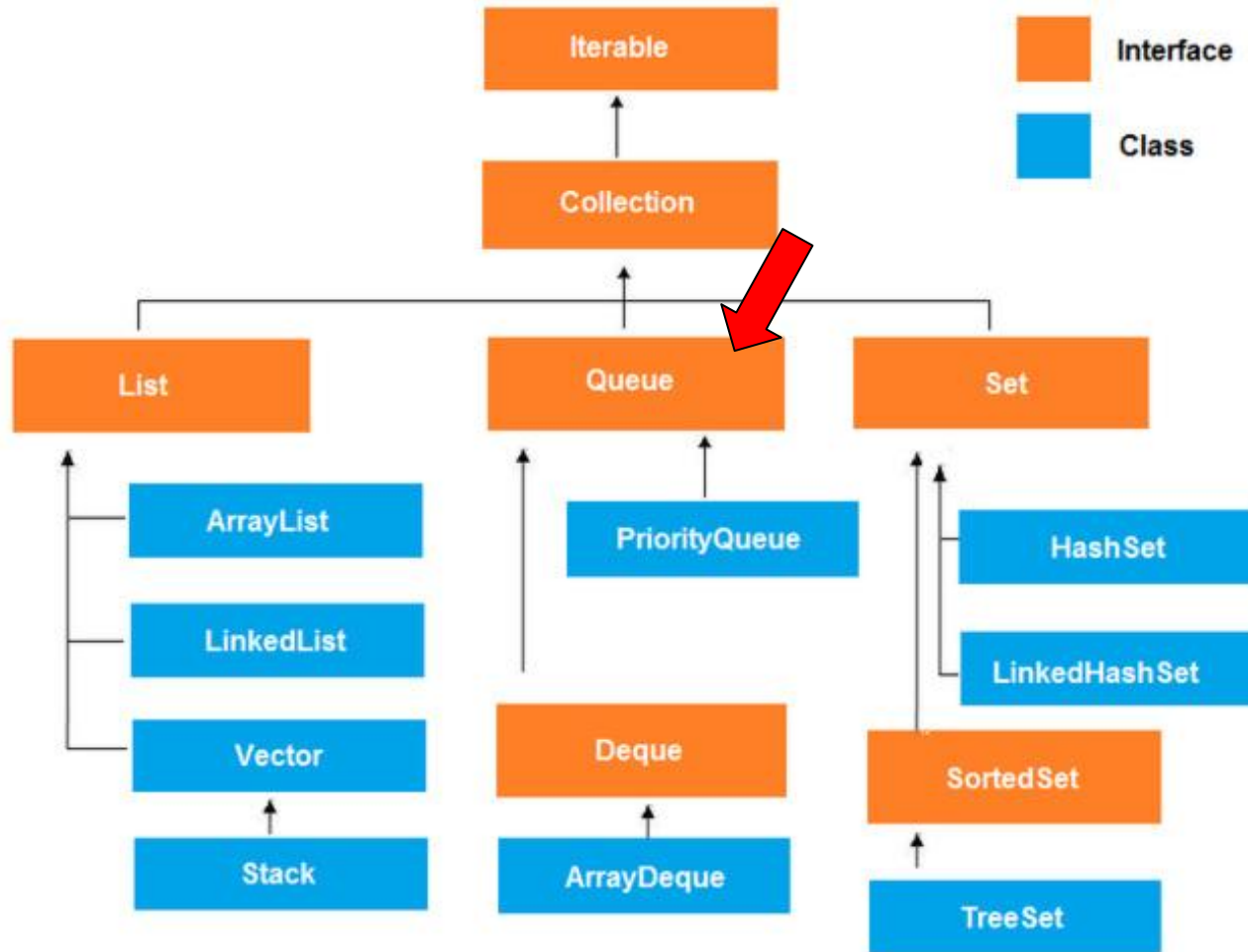
— Programación y Laboratorio III —

Agenda

- **Queue**
- Set
- HashSet
- LinkedHashSet
- TreeSet
- Diagrama de decisiones para uso de colecciones

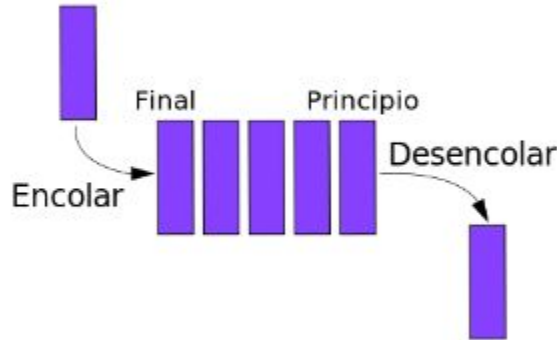


Collection Queue



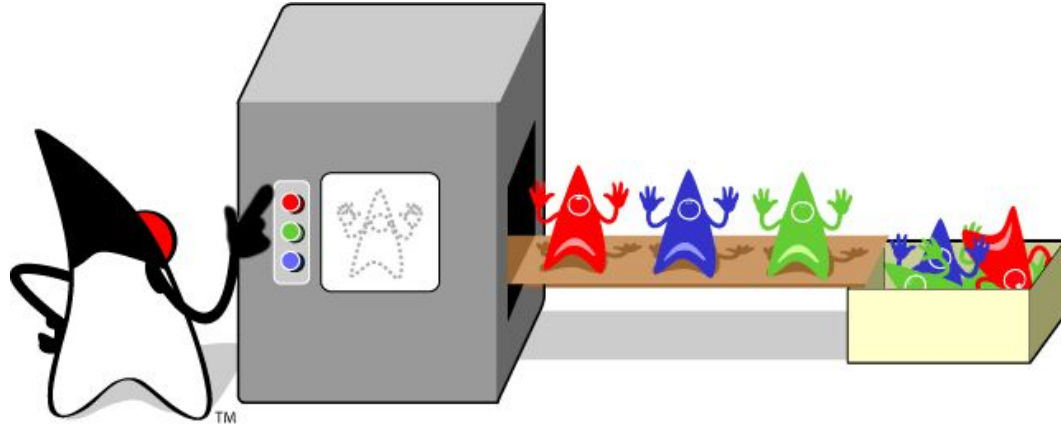
Queue

- Representa al tipo Cola, que es una lista en la que sus elementos se introducen únicamente por un extremo (fin de la cola) y se remueven por el extremo contrario (principio de la cola).

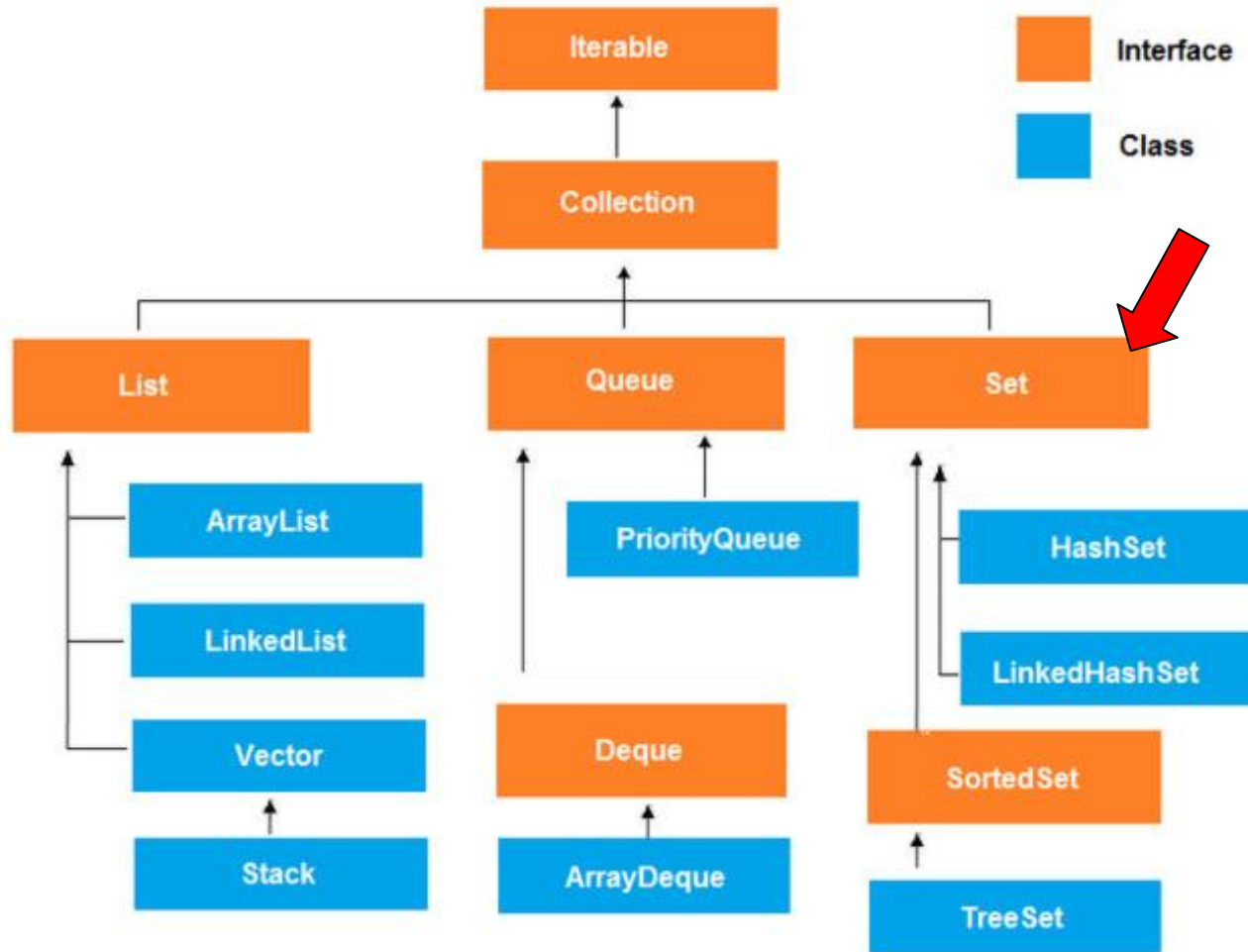


Agenda

- Queue
- **Set**
- HashSet
- LinkedHashSet
- TreeSet
- Diagrama de decisiones para uso de colecciones



Collection Set



Interfaz Set

- Modela los conjuntos de la matemática y sus propiedades.
- Set hereda los métodos de Collection y agrega sus propias restricciones para prohibir el duplicado de elementos.

Si tenemos un objeto que tienen las mismas características (equals) y el mismo hashCode que los objetos que ya se encuentran en el Set →
No se agrega a la colección

Interfaz Set - Algunas operaciones

- `int size();`
- `boolean isEmpty();`
- `boolean contains(Object var1);`
- `Iterator<E> iterator();`
- `boolean add(E var1);`
- `boolean remove(Object var1);`
- **`boolean addAll(Collection<? extends E> var1);`**
- **`boolean retainAll(Collection<?> var1);`**
- **`boolean removeAll(Collection<?> var1);`**

Interfaz Set - Operaciones

- `boolean addAll(Collection<? extends E> var1);`

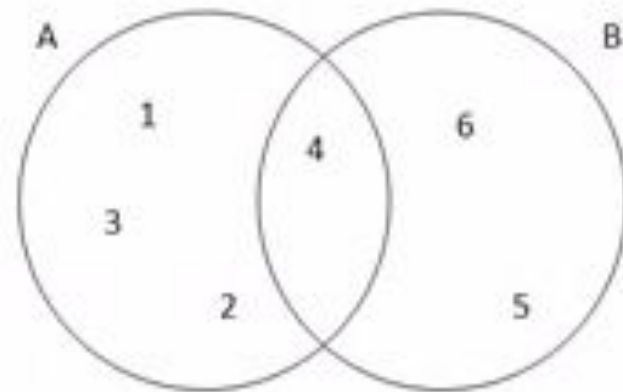
$$A \cup B = \{1,2,3,4,5,6\}$$

- `boolean retainAll(Collection<?> var1);`

$$A \cap B = \{4\}$$

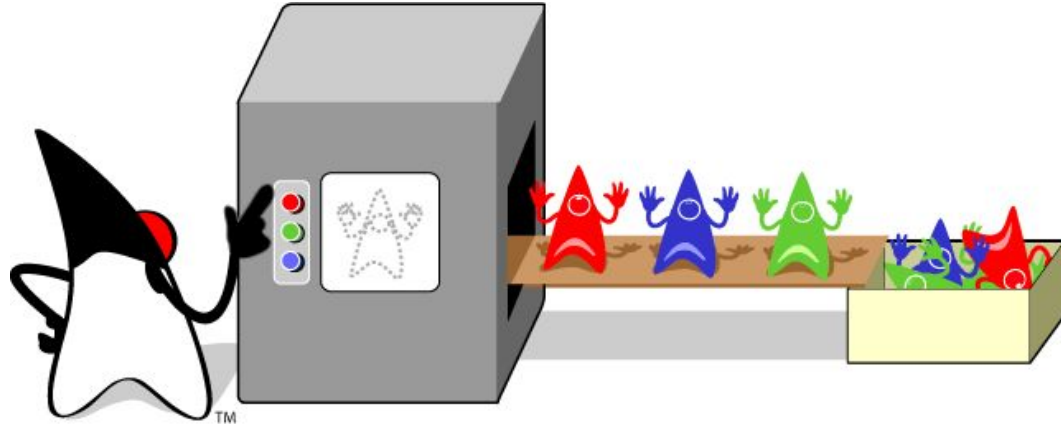
- `boolean removeAll(Collection<?> var1);`

$$A - B = \{1,2,3\}$$



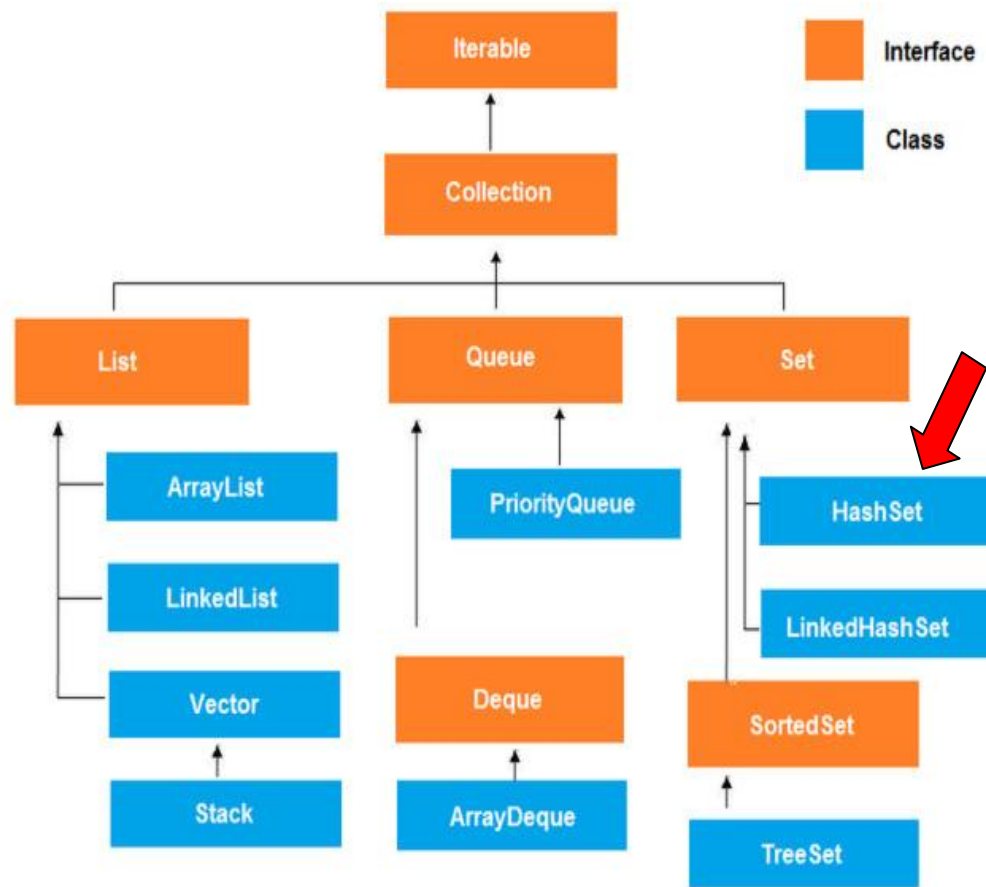
Agenda

- Queue
- Set
- **HashSet**
- LinkedHashSet
- TreeSet
- Diagrama de decisiones para uso de colecciones



HashSet

- Esta implementación de Set almacena los elementos en una **tabla hash**.
- Esta implementación proporciona tiempos constantes en las operaciones básicas siempre y cuando la función hash disperse de forma correcta los elementos dentro de la tabla hash.



HashSet (2)

- Esta clase delega casi todas sus funcionalidades a un **HashMap<T, Object>**, donde el objeto que se inserta en el HashSet será la clave del mapa interno, y se registrará con el valor de un objeto por defecto dentro del mapa.
- Toda la lógica de verificación que el elemento no esté duplicado la maneja el mapa interno utilizado en la instancia del set → Las claves en un HashMap deben ser únicas.

HashSet (3)

1) Creación de HashSet:

```
HashSet<String> hashSet = new HashSet<String>();
```

2) Al llamar al constructor de HashSet, se ejecuta lo siguiente:

```
public HashSet() {  
    map = new HashMap<>();  
}
```

“map” es una variable de instancia de HashSet declarada como:
`HashMap<E, Object> map;`

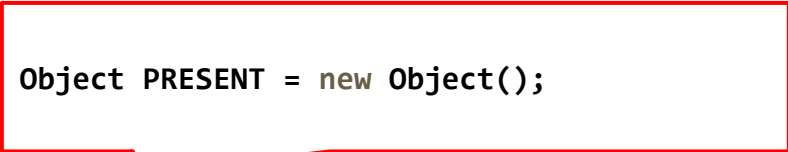
HashSet (4)

3) Una vez creado el HashSet, agregamos un conjunto de nombres:

```
hashSet.add("Pedro");  
hashSet.add("Pepe");  
hashSet.add("Luis");
```

4) Implementación del método add(E e):

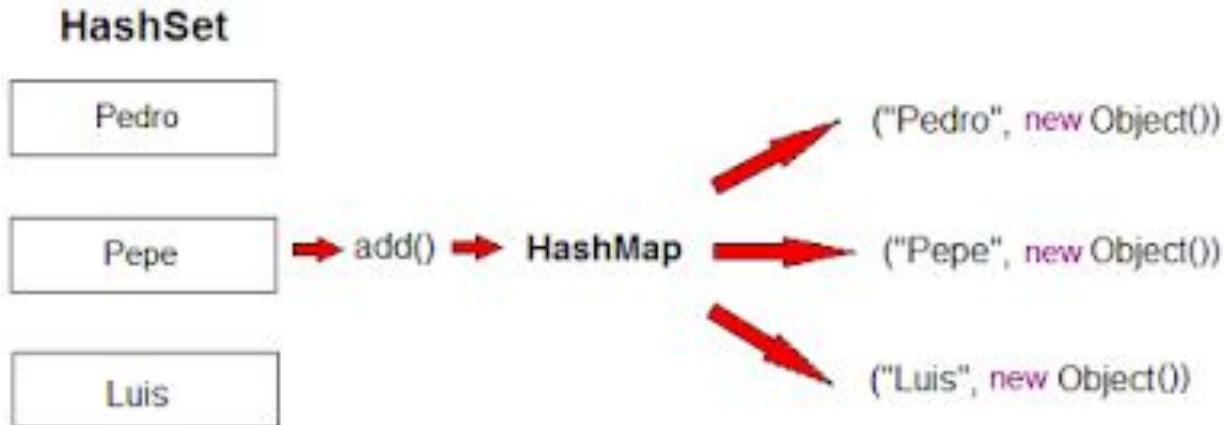
```
public boolean add(E e) {  
    return map.put(e, PRESENT) != null;  
}
```



```
Object PRESENT = new Object();
```

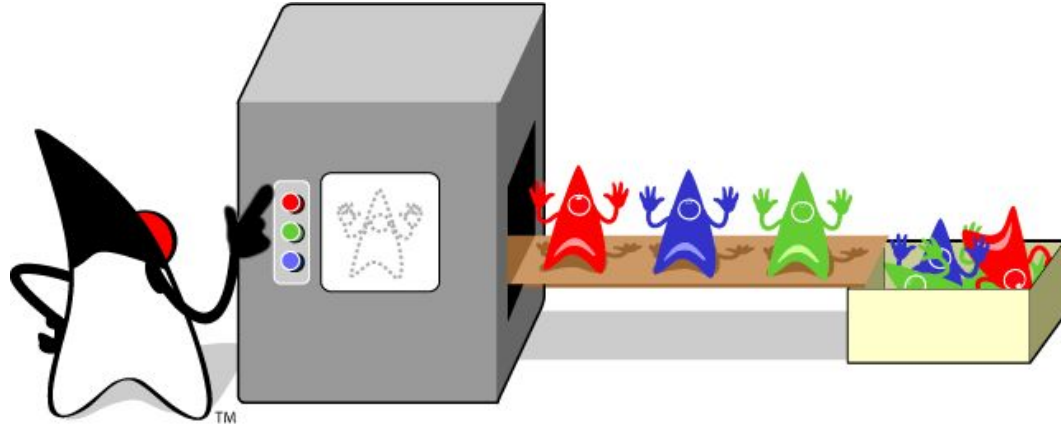
HashSet (4)

- Si ese objeto ya está agregado devuelve el valor anterior de esa key pero si el objeto no está y se agrega al HashMap devuelve null, entonces lo que hace el método `add()` es evaluar que devuelve y así se sabe si agrega o no el objeto.



Agenda

- Queue
- Set
- HashSet
- **LinkedHashSet**
- TreeSet
- Diagrama de decisiones para uso de colecciones

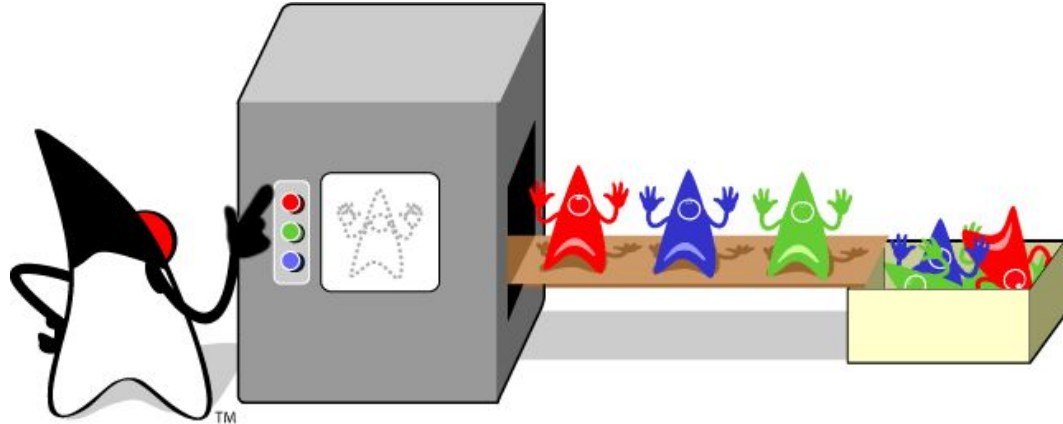


LinkedHashSet

- Es similar a HashSet pero la tabla de dispersión es doblemente enlazada.
- Los elementos que se inserten tendrán enlaces entre ellos. Por lo tanto, las operaciones básicas seguirán teniendo coste constante, con la carga adicional que supone tener que gestionar los enlaces. Sin embargo habrá una mejora en la iteración, ya que al establecerse enlaces entre los elementos no tendremos que recorrer todas las entradas de la tabla, el coste sólo estará en función del número de elementos insertados.
- En este caso, al haber enlaces entre los elementos, estos enlaces definirán el **orden** en el que se insertaron en el conjunto, por lo que el orden de iteración será el mismo orden en el que se insertaron.

Agenda

- Queue
- Set
- HashSet
- ~~LinkedHashSet~~
- **TreeSet**
- Diagrama de decisiones para uso de colecciones



TreeSet

- 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**.
- Garantiza, siempre, un rendimiento de $\log(N)$ en las operaciones básicas, debido a la estructura de árbol empleada para almacenar los elementos.

Agenda

- Queue
- Set
- HashSet
- LinkedHashSet
- TreeSet
- Diagrama de decisiones para uso de colecciones

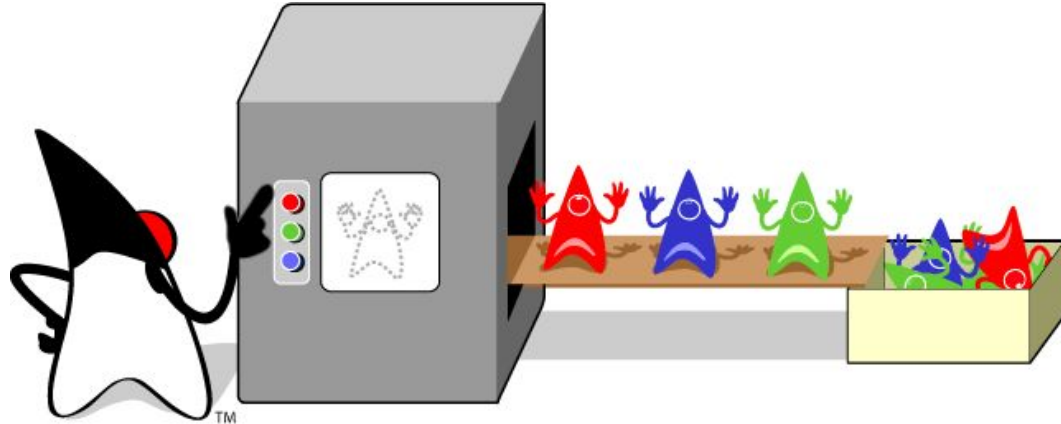
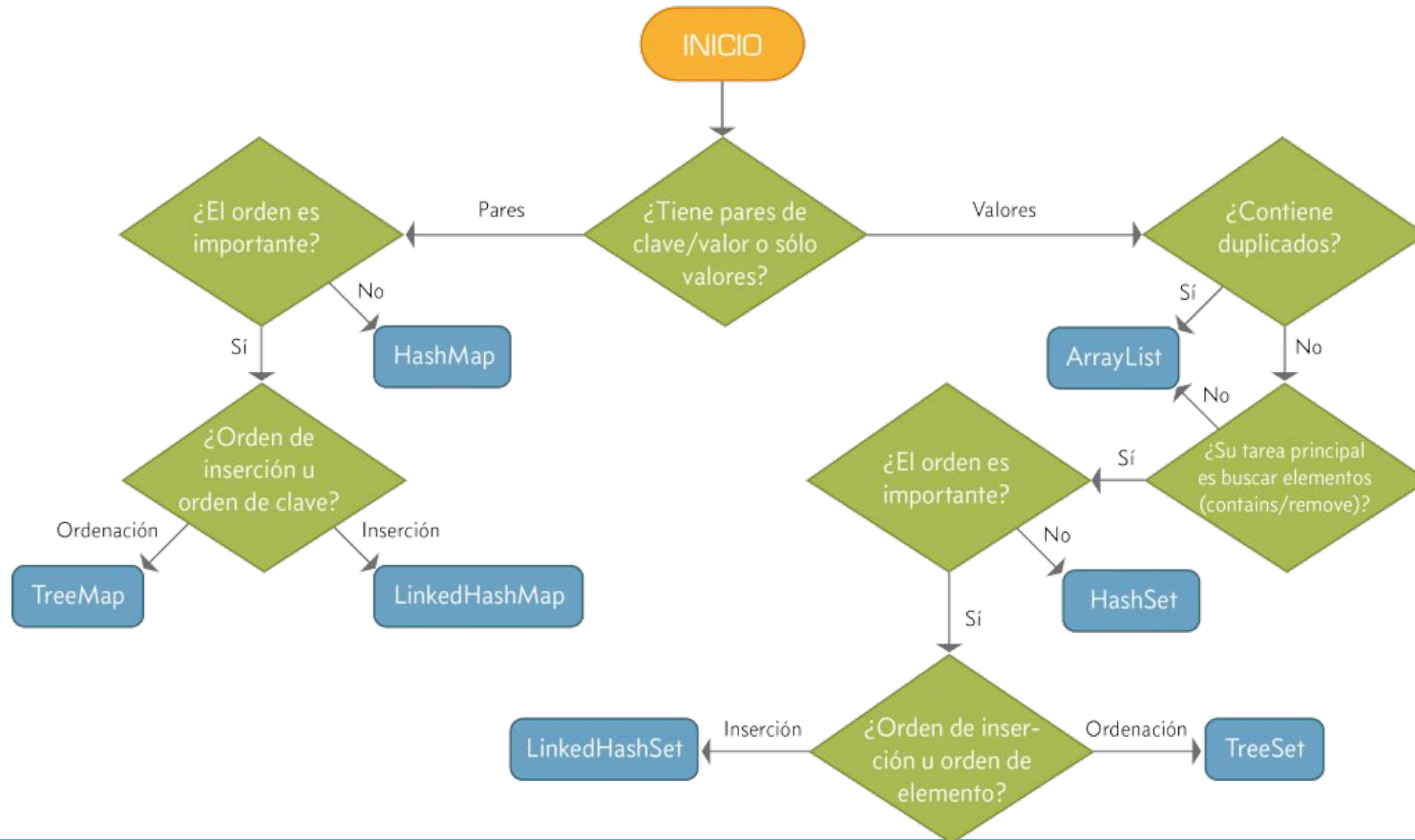


Diagrama de decisión para uso de colecciones Java



Bibliografía oficial

- <https://docs.oracle.com/javase/tutorial/collections/implementations/set.html>
- <https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html>
- <https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html>
- <https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashSet.html>
- <https://docs.oracle.com/javase/tutorial/collections/implementations/summary.html>