Java Colecciones

Octavio Robleto

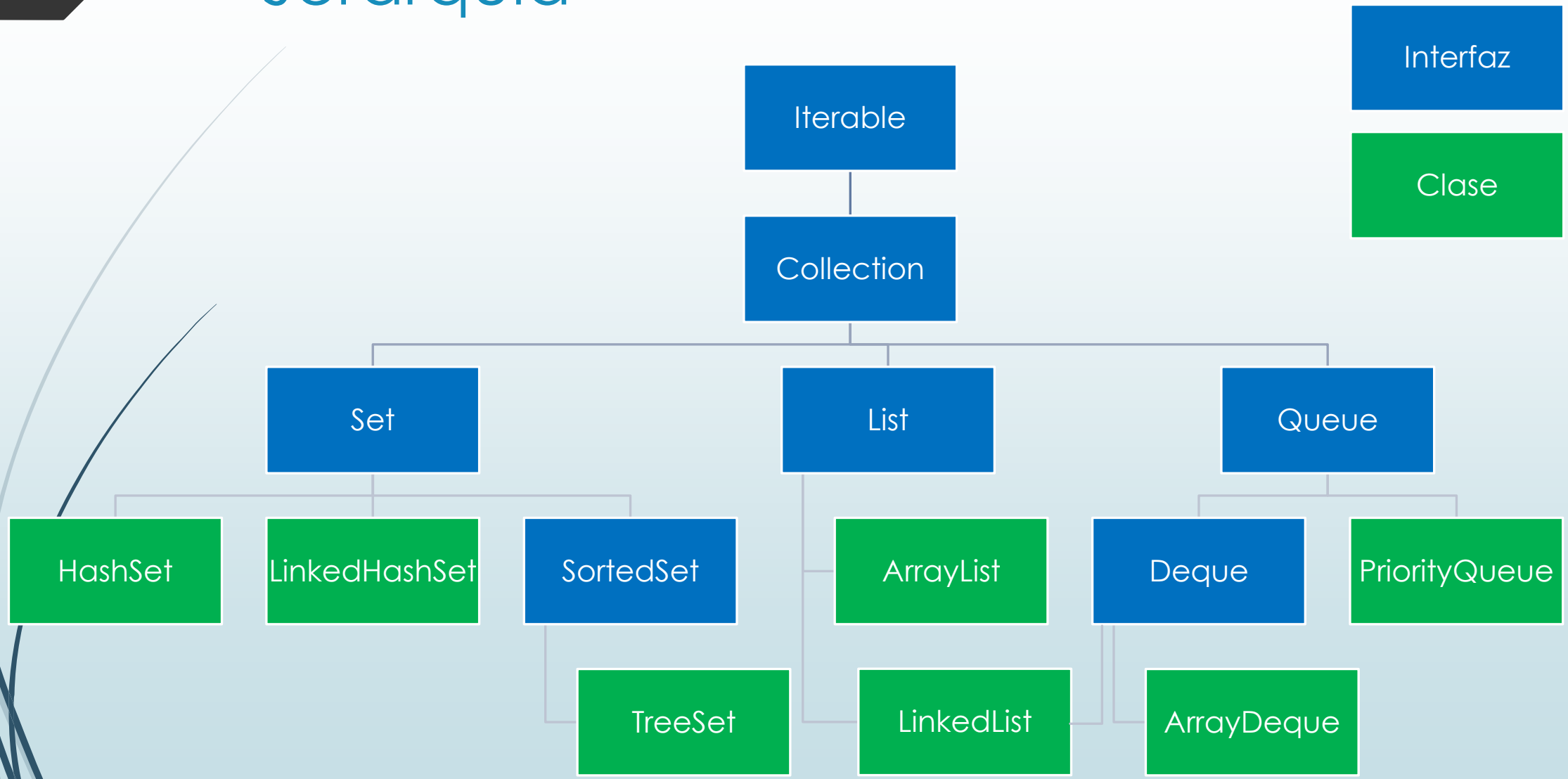
octavio.robleto@gmail.com

https://octaviorobleto.com

Que son?

- Conjunto de objetos que se pueden agrupar, por lo que se pueden manipular y recorrer si es necesario.
- A diferencia de los arreglos convencionales "**int[] diaNumeros = new int[7];**"estos son dinámicos, por lo que podemos insertar los objetos necesarios sin indicar el tamaño único e inicial.

Jerarquía


Interfaz
Clase
Iterable
Collection
Set
List
Queue
HashSet
LinkedHashSet
SortedSet
ArrayList
Deque
PriorityQueue
TreeSet
LinkedList
ArrayDeque

Métodos mas usados en la Interfaz

Tipo	Método y descripción
boolean	**add**(E e)Asegura que esta colección contiene el elemento especificado (operación opcional).
boolean	**addAll**(Collection<? extends E> c)Agrega todos los elementos de la colección especificada a esta colección (operación opcional).
void	**clear**()Elimina todos los elementos de esta colección (operación opcional).
boolean	**contains**(Object o)Devuelve verdadero si esta colección contiene el elemento especificado.
boolean	**containsAll**(Collection<?> c)Devuelve verdadero si esta colección contiene todos los elementos de la colección especificada.
boolean	**equals**(Object o)Compara el objeto especificado con esta colección para la igualdad.
int	**hashCode**()Devuelve el valor del código hash para esta colección.
boolean	**isEmpty**()Devuelve verdadero si esta colección no contiene elementos.
Iterator<E>	**iterator**()Devuelve un iterador sobre los elementos de esta colección.
boolean	**remove**(Object o)Elimina una sola instancia del elemento especificado de esta colección, si está presente (operación opcional).
int	**size**()Devuelve el número de elementos en esta colección.

Interfaz Iterator

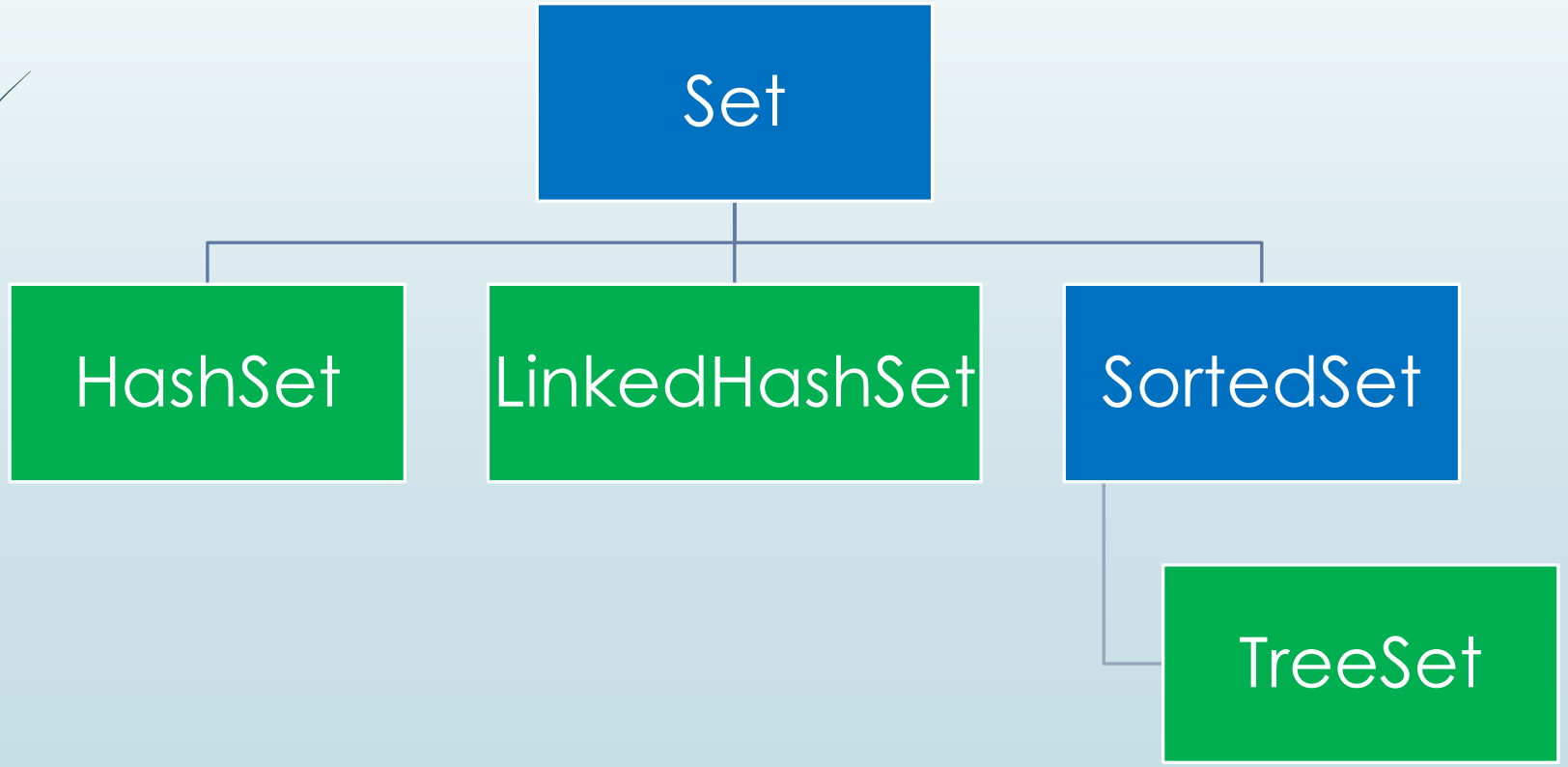
- Algo particular de estas colecciones es que no se pueden recorrer y operar al mismo tiempo, para eso cada clase en las colecciones implementan un método que retorna un iterador.

- Si solo deseamos acceder al elemento sin manipularlo podemos usar el for-each o for mejorado.

Tipo	Método y descripción
boolean	**hasNext**()Devuelve truesi la iteración tiene más elementos.
E	**next**()Devuelve el siguiente elemento en la iteración.
void	**remove**()Elimina de la colección subyacente el último elemento devuelto por este iterador (operación opcional).

Interfaz Set

- Esta interfaz tiene una peculiaridad, y es que no admite duplicados y es especialmente útil para ir almacenando datos sin la preocupación de que alguno se repita. Estos elementos o datos pueden estar ordenados o no.



HashSet

- Este implementación almacena los elementos en una tabla *hash*.
- Representa un conjunto de valores únicos sin ordenar, no puede tener valores duplicados y tiene una iteración más rápida que otras colecciones.

```
//Declarar e Intanciar un HashSet
HashSet<String> miHashSet = new HashSet<>();
//Declarar e Intanciar unsegundo HashSet
Set<Integer> miHashSet2 = new HashSet<>();
```

LinkedHashSet

- Esta implementación almacena los elementos en función del orden de inserción. Es, simplemente, un poco más costosa que **HashSet.**

- Define el concepto de conjunto añadiendo una lista doblemente enlazada en la ecuación que nos asegura que los elementos siempre se recorren de la misma forma.

```
Set<String> miLinkedHashSet = new LinkedHashSet<>();
```

TreeSet

- Esta implementación almacena los elementos ordenándolos en función de sus valores (Implementando el algoritmo del **árbol rojo - negro**). Es bastante más lento que **HashSet**.

```
Set<String> miTreeSet = new TreeSet<>();
```

¿Y cómo sabe que los objetos son iguales?

- **equals** es el más sencillo de entender ya que comprueba si los dos objetos son del mismo tipo y si su nombre coincide. En tal caso el resultado será true y en todos los demás false. Para ello el método realiza una serie de comparaciones.

Clase Auto

```
@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (getClass() != obj.getClass()) {
        return false;
    }
    final Auto autoAux = (Auto) obj;
    if (!Objects.equals(this.patente, autoAux.patente)) {
        return false;
    }
    return true;
}
```

¿Y cómo sabe que los objetos son iguales?

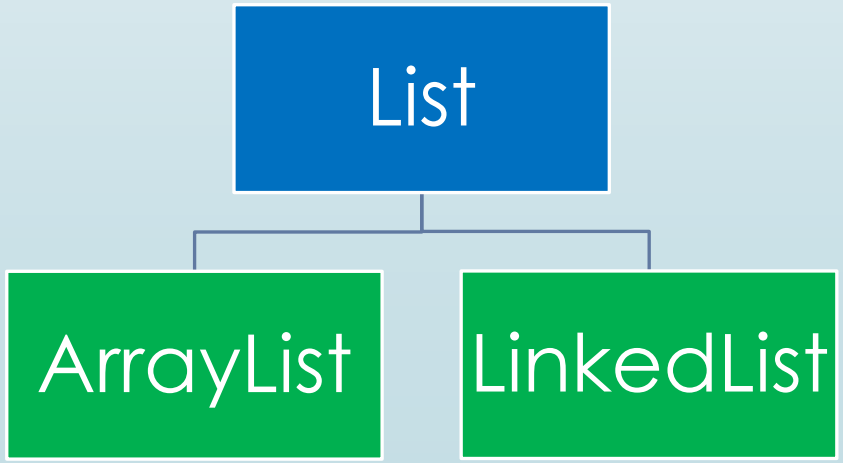
- **hashcode** este método viene a complementar al método equals y sirve para comparar objetos de una forma más rápida en estructuras Hash ya que únicamente nos devuelve un número entero

Clase Auto

```
@Override
public int hashCode() {
    int hash = 7;
    hash = 71 * hash + Objects.hashCode(this.patente);
    return hash;
}
```

Interfaz List

- Es la colección mas usada y a diferencia de la interfaz set, List permite elementos duplicados y acceder a los elementos por medio de índices.
- A parte de los métodos heredados de Collection, añade métodos que permiten mejorar los siguientes puntos:
  - **Acceso posicional a elementos:** manipula elementos en función de su posición en la lista.
  - **Búsqueda de elementos:** busca un elemento concreto de la lista y devuelve su posición.
  - **Iteración sobre elementos:** mejora el Iterator por defecto.



Métodos mas usados para esta Interfaz

Tipo	Método y descripción
E	**get**(int index) Devuelve el elemento en la posición especificada en esta lista.
E	**remove**(int index) Elimina el elemento en la posición especificada en esta lista (operación opcional).
E	**set**(int index, E element) Reemplaza el elemento en la posición especificada en esta lista con el elemento especificado (operación opcional).
E	**add**(int index, E element) Inserta el elemento especificado en la posición especificada en esta lista (operación opcional).

ArrayList

- Esta es la implementación típica. Se basa en un array redimensionable que aumenta su tamaño según crece la colección de elementos. Es la que mejor rendimiento tiene sobre la mayoría de situaciones.

```
List<String> miArrayList = new ArrayList();
```

LinkedList

- Esta implementación se basa en una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento.

```
List<String> miLista = new LinkedList();
```

Interfaz ListIterator

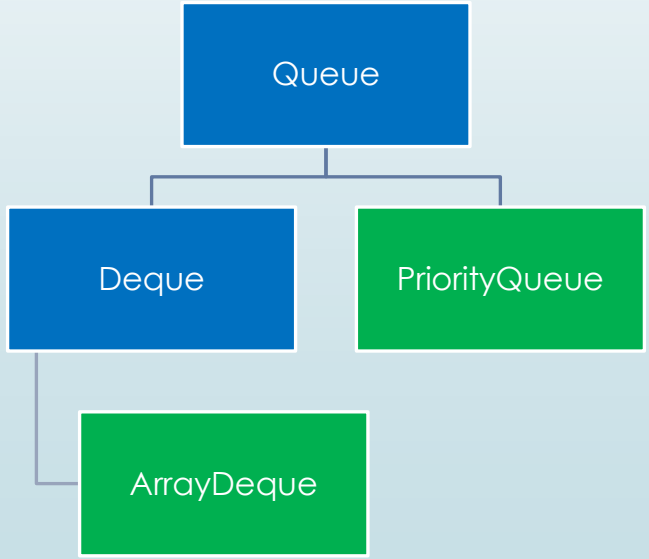
- Al igual que todas las colecciones necesitamos un iterador para poder manipular y recorrer al mismo tiempo.
- Para las linkedList hay mas métodos.

Tipo	Método y descripción
void	**add**(E e)Inserta el elemento especificado en la lista (operación opcional).
boolean	**hasNext**()Devuelve truesi este iterador de lista tiene más elementos al recorrer la lista en la dirección hacia adelante.
boolean	**hasPrevious**()Devuelve truesi este iterador de lista tiene más elementos al recorrer la lista en la dirección inversa.
E	**next**()Devuelve el siguiente elemento de la lista y avanza la posición del cursor.
int	**nextIndex**()Devuelve el índice del elemento que sería devuelto por una llamada posterior a next().
E	**previous**()Devuelve el elemento anterior en la lista y mueve la posición del cursor hacia atrás.int
void	**remove**()Elimina de la lista el último elemento que fue devuelto por next()o previous()(operación opcional).
void	**set**(E e)Reemplaza el último elemento devuelto por next()o previous()con el elemento especificado (operación opcional).

```
ListIterator<String>  miIterador = miLista.listIterator();
```

La interfaz Queue

- Es una interfaz que extiende a Collection y proporciona operaciones para trabajar con una cola.
- Se utiliza para mantener los elementos a punto de ser procesados y proporciona varias operaciones como la inserción, eliminación, etc



ArrayDeque

- Este es un tipo especial permite agregar o eliminar un elemento de ambos lados de la cola.

Tipo	Método y descripción
E	**peekFirst**() Recupera, pero no elimina, el primer elemento de esta deque, o devuelve nulo si esta deque está vacía.
E	**peekLast**() Recupera, pero no elimina, el último elemento de esta deque, o devuelve nulo si esta deque está vacía.
E	**pollFirst**() Recupera y elimina el primer elemento de esta deque, o devuelve nulo si esta deque está vacía.
E	**pollLast**() Recupera y elimina el último elemento de esta deque, o devuelve nulo si esta deque está vacía.

```
Deque<Integer> cola = new ArrayDeque<Integer>();
```

PriorityQueue

- Una variante de una cola clásica la implementa esta clase. Cuando se agregan elementos a la cola se organiza según su valor, por ejemplo si es un número se ingresan de menor a mayor.

Tipo	Método y descripción
E	**peek**() Recupera, pero no elimina, el primer elemento de esta PriorityQueue, o devuelve nulo si esta PriorityQueue está vacía.
E	**poll**() Recupera y elimina el primer elemento de esta PriorityQueue, o devuelve nulo si esta PriorityQueue está vacía.

```
Queue<Integer> colaPrioritaria = new PriorityQueue<Integer>();
```