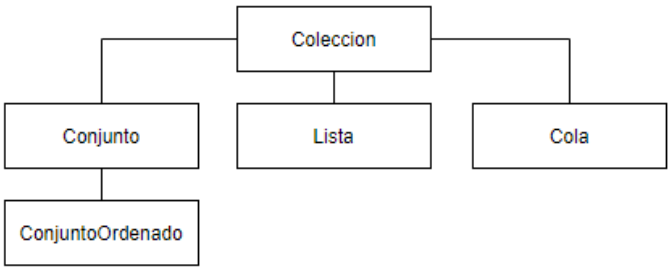
COLECCIÓN DE DATOS

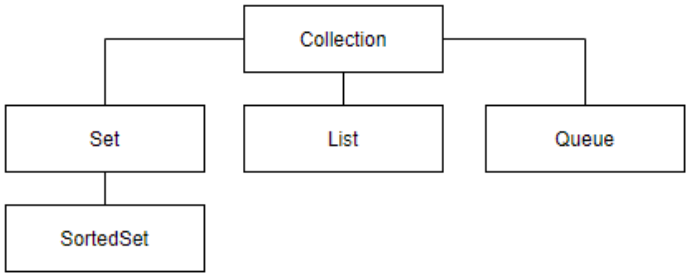
1. ¿Qué es una colección en Java?

Una **colección** es un **objeto** que **recopila** y **organiza** otros **objetos**. Define las formas específicas con las que se puede acceder y con las que se pueden gestionar estos objetos, denominados **elementos de la colección**.

Frente a un Array que hemos visto anteriormente, la colección recopilación dinámica, es decir, que puede aumentar/disminuir su tamaño de forma dinámica y no están limitados a un tamaño concreto y fijo; a parte de proporcionar diversas funcionalidades en función del uso al que va destinado.

Por lo tanto, Java nos proporciona un amplio conjunto de clases. Estas clases, que representan colecciones en Java, se encuentran dentro del paquete **java.util**. Y se basan en una serie de interfaces que definen los métodos necesarios para su gestión y que estas clases implementarán.

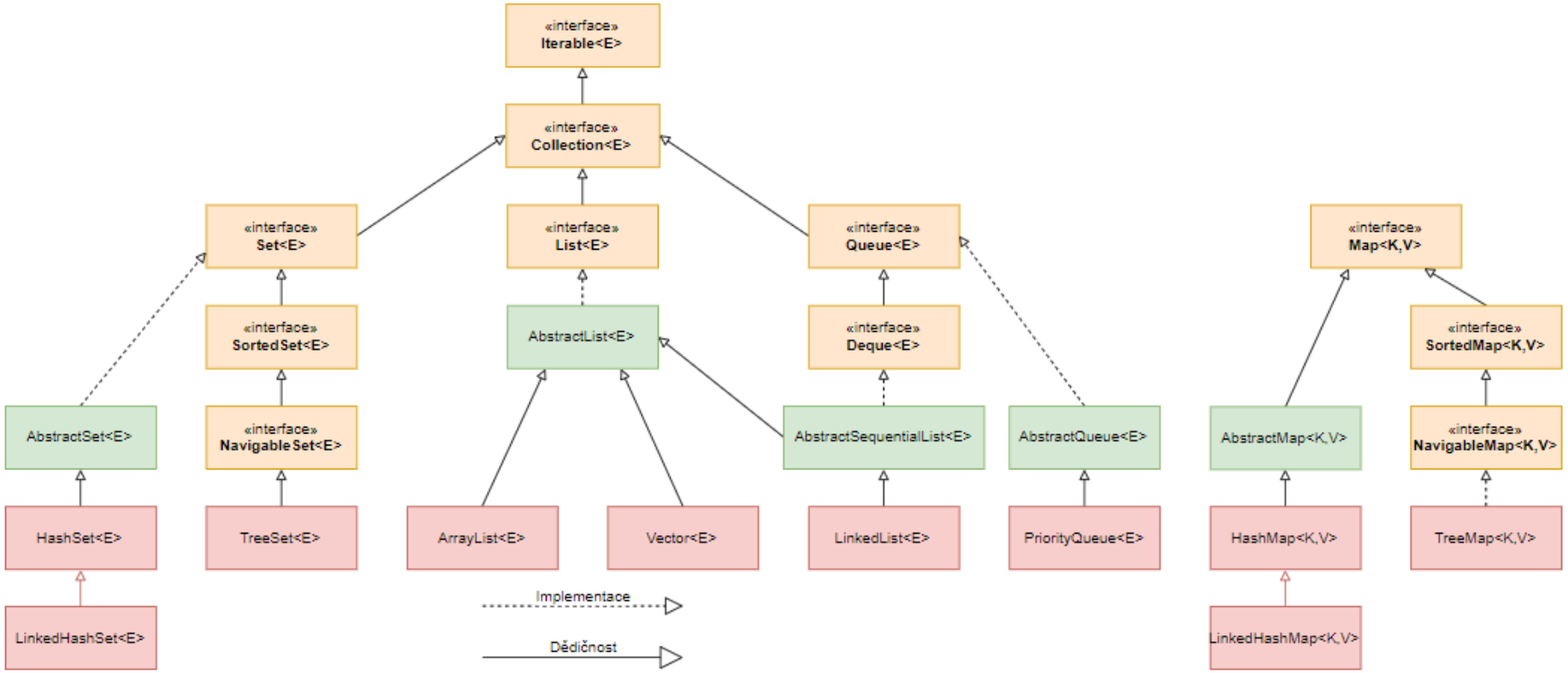




Indicar que a su vez las interfaces de las colecciones se basan en genéricos, lo que permite crear colecciones que resulten fáciles de utilizar con múltiples tipos de datos definidos por el usuario según sus necesidades.

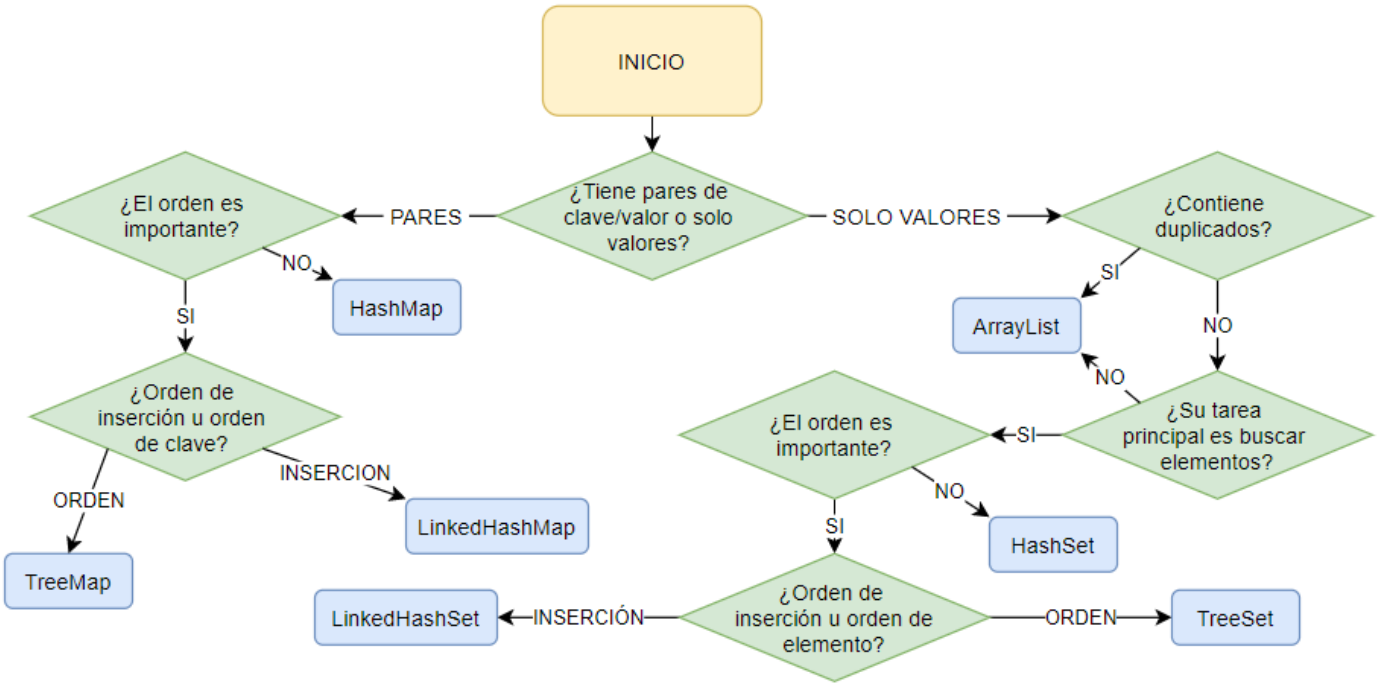
- **Interfaz LIST**: Una colección **List** (lista) debe almacenar los elementos en la misma forma en la que fueron insertados.
- **Interfaz SET**: La interfaz Set define una colección que no puede contener elementos duplicados. Esta interfaz contiene, únicamente, los métodos heredados de Collection añadiendo la restricción de que los elementos duplicados están prohibidos.
- **Interfaz QUEUE**: Una colección **Queue** (cola) produce los elementos en un orden determinado por una disciplina de cola.
- **Interfaz Map**. La Interface Map (java.io.Map) en Java, nos permite representar una estructura de datos para almacenar pares "clave/valor"; de tal manera que para una clave solamente tenemos un valor.

De estas interfaces derivan una serie de clases que conforman las opciones que Java nos brinda a la hora de crear y utilizar colecciones de datos u objetos.


«interface»
Iterable<E>
«interface»
Collection<E>
«interface»
Set<E>
«interface»
List<E>
«interface»
Queue<E>
«interface»
Map<K,V>
«interface»
SortedSet<E>
AbstractList<E>
«interface»
Deque<E>
«interface»
SortedMap<K,V>
AbstractSet<E>
«interface»
NavigableSet<E>
AbstractSequentialList<E>
AbstractQueue<E>
AbstractMap<K,V>
«interface»
NavigableMap<K,V>
HashSet<E>
TreeSet<E>
ArrayList<E>
Vector<E>
LinkedList<E>
PriorityQueue<E>
HashMap<K,V>
TreeMap<K,V>
LinkedHashSet<E>
LinkedHashMap<K,V>
Implementace
Dědičnost

2. ¿Cuándo utilizar un tipo de colección u otro?

Depende principalmente del objetivo al que va a destinado y las características/premisas requeridas. A modo de guía general se plantea el siguiente diagrama:



De entre las clases más utilizadas y que explicaremos en mayor detalle en esta unidad están: ArrayList y HashMap.

3. INTERFAZ COLECCIÓN:

Las colecciones representan grupos de objetos, denominados elementos. Esta interfaz es la más genérica, refiriéndose a cualquier tipo de colección que contenga elementos.

En esta interfaz se definen números métodos que utilizan las restantes clases hijas:

- **boolean add(Object o).** Añade un elemento (objeto) a la colección.
- **boolean remove(Object o).** Elimina un determinado elemento (objeto) de la colección
- **void clear().** Elimina todos los elementos de la colección.
- **boolean remove(Object o).** Elimina un determinado elemento (objeto) de la colección.
- **boolean contains(Object o).** Indica si la colección contiene el elemento (objeto) indicado.
- **boolean isEmpty().** Indica si la colección está vacía (no tiene ningún elemento).
- **int size().**Nos devuelve el número de elementos que contiene la colección
- **Iterator iterator().** Proporciona un iterador para acceder a los elementos de la colección.
- **Object [] toArray().** Nos devuelve la colección de elementos como un array de objetos.

4. USO ARRAYLIST:

4.1. CREAR ARRAYLIST.

```
import java.util.ArrayList;
ArrayList<String> cars = new ArrayList<String>();
```

4.2. AÑADIR UN ELEMENTO

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Volvo");
cars.add("BMW");
cars.add("Ford");
System.out.println(cars);
```

4.3. CAMBIAR UN ELEMENTO.

```
cars.set(0, "Opel");
```

4.4. ELIMINAR UN ELEMENTO.

```
cars.remove(0);
```

4.4.1. ELIMINAR TODOS LOS ELEMENTOS

```
cars.remove(0);
```

4.5. OBTENER EL TAMAÑO DE UN ARRAYLIST – NÚMERO DE ELEMENTOS INSERTADOS.

```
cars.size();
```

4.6. VER SI DENTRO DE LA COLECCIÓN HAY UN VALOR DETERMINADO.

```
System.out.println(cars.contains("Opel"));
```

4.7. OBTENER INDICE DADO UN ELEMENTO.

```
ArrayList<String> cars = new ArrayList<String>();
cars.add("Opel");       // pos 0
cars.add("Mazda");      // pos 1
cars.add("BMW");        // pos 2
cars.add("Mercedes");   // pos 3
cars.add("Nissan");     // pos 4
cars.add("Mazda");      // pos 5

System.out.println(cars.indexOf("BMW"));        // Retorna: 2.
System.out.println(cars.indexOf("Mazda"));      // Retorna: 1, el primer elemento que encuentra.
System.out.println(cars.lastIndexOf("Mazda")); // Retorna: 5, el último elemento que encuentra.
System.out.println(cars.indexOf("Ford"));       // Retorna: -1, pues no encuentra ese elemento.
```

4.8. RECORRER UN ARRAYLIST.

4.8.1. FOR

```
for (int i = 0; i < cars.size(); i++) {
    String element = cars.get(i);
    System.out.println(element);
}
```

4.8.2. FOREACH

```
for (String element : cars) {
    System.out.println(element);
}
```

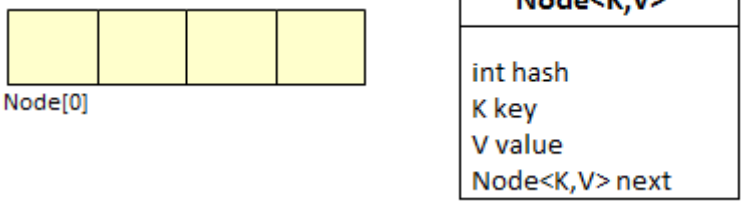
5. USO DE HASHMAP

Como hemos visto en el apartado anterior, para acceder/modificar a un elemento de una colección ArrayList es necesario el índice de la lista sobre que el que actuar: **indice** (tipo int).

En cambio, un HashMap almacena los artículos en pares **"key/value",** y puedes acceder al dato a través de la "key" que no tiene que ser un número (int) sino cualquier otro tipo (ej. String).

Se utiliza un tipo de variable como clave (índice) de otro objeto (valor).

Internamente Hashmap consiste en array de nodos, donde un nodo es una clase con 4 atributos:



Además, aplica un mecanismo de **Hashing**. Hashing es un proceso mediante el cual convertir un objeto a un formato número utilizando el método "hashCode()". Pero estas operaciones de conversión las realiza de forma interna y transparente al programador.

5.1. CREAR HASHMAP

```
import java.util.HashMap; // import the HashMap class
HashMap<String, String> capitalCities = new HashMap<String, String>();
```

5.2. ANADIR ELEMENTO / MODIFICAR ELEMENTO.

Añade un nuevo elemento (par key/value) en caso de que el key ya existe actualiza su valor.

```
HashMap<String, String> capitalCities = new HashMap<String, String>();
// Add keys and values (Country, City)
capitalCities.put("England", "London");
capitalCities.put("Germany", "Berlin");
capitalCities.put("Norway", "Oslo");
System.out.println(capitalCities);
```

5.3. ACCEDER A UN ELEMENTO. Se hace a través del Key.

```
capitalCities.get("England");
```

5.4. ELIMINAR UN ELEMENTO.

```
capitalCities.remove("England");
```

5.4.1. ELIMINAR TODOS LOS ELEMENTOS

```
capitalCities.clear(); // Borrar todos los registros.
```

5.5. OBTENER TAMAÑO.

```
capitalCities.size();
```

5.6. COMPROBAR SI LA LISTA CONTIENE UNA CLAVE DETERMINADA.

```
System.out.println(capitalCities.containsKey("Madrid"));
```

5.7. RECORRER HASHMAP.

```
for (String key : capitalCities.keySet()) {
    System.out.println("key: "+key+ " & value: "+capitalCities.get(key));
}

for (Entry<String, Integer> entry : capitalCities.entrySet()) {
    System.out.println("key: "+entry.getKey()+ " value: "+entry.getValue());
}
```

6. JAVA ITERATOR.

Una de las situaciones más habituales en la programación es recorrer una lista de elementos. En Java se puede realizar de varias maneras, dos de las más comunes son **usando Java Iterator o usando un bucle forEach**.

6.1. EJEMPLO USO DE JAVA ITERATOR:

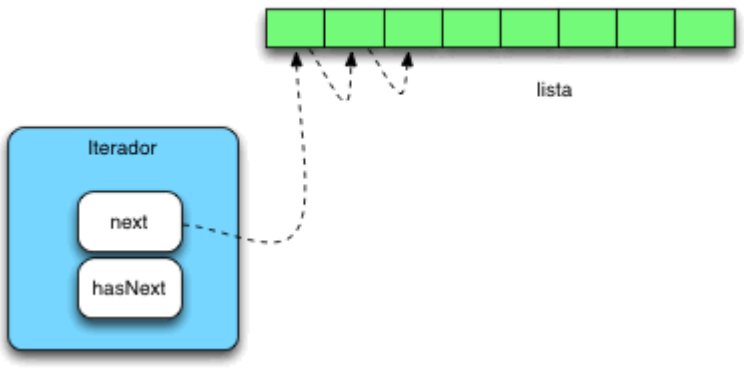
```
public class Ejemplo_iteradores {
    public static void main(String[] args) {
        //Creamos un arrayList
        ArrayList<String> lista;
        lista = new ArrayList<>();
        lista.add("Elemento 1");
        lista.add("Elemento 2");
        lista.add("Elemento 3");
        lista.add("Elemento 4");
        lista.add("Elemento 5");

        //Creamos un iterador
        Iterator<String> iterador = lista.iterator();

        //Usamos el método hasNext, para comprobar si hay algun elemento
        while(iterador.hasNext()){
            //El iterador devuelve el proximo elemento
            System.out.println(iterador.next());
        }
    }
}
```

Un iterador es un objeto que nos permite recorrer una lista y presentar por pantalla todos sus elementos. **Dispone de dos métodos clave para realizar esta operación hasNext() y next().**

```
1. while (it.hasNext()) {
2.
3. System.out.println(it.next());
4.
5. }
```



7. Bucle Iterator vs bucle foreach

A partir de Java 5 existe otra forma de recorrer una lista que es mucho más cómoda y compacta, el uso de bucles foreach. **Un bucle foreach se parece mucho a un bucle for con la diferencia de que no hace falta una variable i de inicialización**:

Por lo general, es recomendable “foreach” por sencillez, obtenido un resultado claramente superior y más legible.

```
1. for (String nombre : lista) {
2. 
3. System.out.println(nombre);
4. }
```