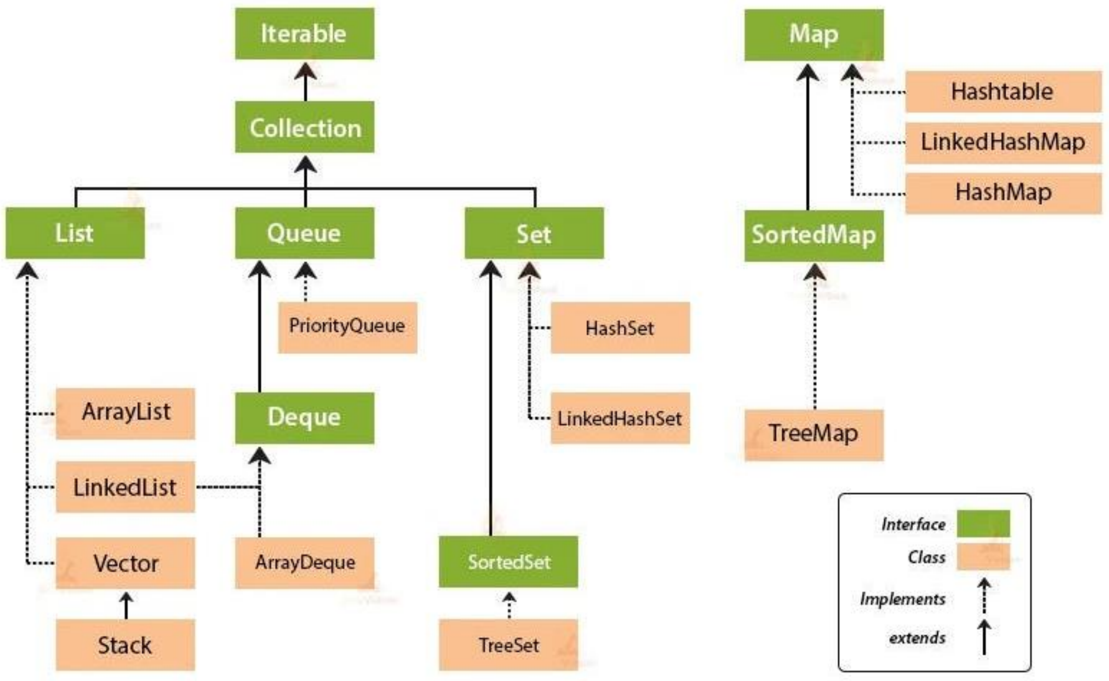
Academia Xideral

Regina Rodríguez Campo Garrido

A collection is a group of objects contained within a single object. Thanks to collections, we can store any type of object and use a series of common methods. We can better understand collections with the following mind map:



List

List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

- **Arraylist:** It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized.

```
List<String> ArrayList = new ArrayList<>();
ArrayList.add("Apple");
ArrayList.add("Banana");
System.out.println(ArrayList);
```

- **LinkedList:** It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized.

```
List<String> LinkedList = new LinkedList<>();
LinkedList.add("Apple");
LinkedList.add("Banana");
System.out.println(LinkedList);
```

Queue

Queue interface maintains the first-in-first-out order. It can be defined as an ordered list that is used to hold the elements which are about to be processed.

- **PriorityQueue:** It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

```
PriorityQueue<Integer> queue = new PriorityQueue<>();

queue.add(5);
queue.add(1);
queue.add(3);
queue.add(2);

System.out.println("Element with highest priority: " + queue.poll());
System.out.println("Remaining elements in the queue: " + queue);
```

- **Deque:** In Deque, we can remove and add the elements from both the side of the list.

```
Deque<String> deque = new ArrayDeque<>();

deque.addFirst("First");
deque.addLast("Last");
deque.addLast("Another Last");

System.out.println("Removed from front: " + deque.removeFirst());
System.out.println("Removed from back: " + deque.removeLast());
```

Set

It represents the unordered set of elements which doesn't allow us to store the duplicate items.

- **HashSet:** It allows to store unique items and access them in constant time (on average). No duplicate values are stored.

```
Set<String> set = new HashSet<>();

set.add("Apple");
set.add("Banana");
System.out.println("HashSet contents: " + set);
System.out.println("Contains 'Apple': " + set.contains("Apple"));

set.remove("Banana");
System.out.println("HashSet after removal: " + set);
```

- **LinkedHashSet:** Uses a doubly-linked list to maintain the order of elements as they were inserted, allowing iteration through the elements in the order of insertion.

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

LinkedHashSet.add("Apple");
LinkedHashSet.add("Banana");
LinkedHashSet.add("Cherry");
System.out.println("LinkedHashSet contents: " + LinkedHashSet);
set.remove("Cherry");
System.out.println("LinkedHashSet after removal: " + LinkedHashSet);
```

- **TreeSet:** The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided.

```
Set<Integer> TreeSet = new TreeSet<>();

TreeSet.add(5);
TreeSet.add(1);
TreeSet.add(3);
TreeSet.add(2);
System.out.println("TreeSet contents: " + TreeSet);
set.remove(2);
System.out.println("TreeSet after removal: " + TreeSet);
```

Map

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

- **HashMap:** It provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs.

```
Map<String, Integer> map = new HashMap<>();

map.put("Apple", 3);
map.put("Banana", 2);
System.out.println("HashMap: " + map);
map.remove("Banana");
System.out.println("HashMap after removal: " + map);
```

- **TreeMap:** Is used to implement the Map interface and NavigableMap along with the Abstract Class. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.

```
Map<String, Integer> TreeMap = new TreeMap<>();

TreeMap.put("Apple", 3);
TreeMap.put("Banana", 2);
TreeMap.put("Cherry", 5);
System.out.println("TreeMap: " + TreeMap);
System.out.println("Apple count: " + TreeMap.get("Apple"));
map.remove("Banana");
System.out.println("TreeMap after removal: " + TreeMap);
```